Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

(2)

REPORT DOCUMENTATION PAGE

AD-A214 786

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
DULE	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
BER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: The Johns Hopkins University / 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: Air Force Office of Scientific Research /NM
6c. ADDRESS (City, State, and ZIP Code): Charles and 34th Streets, Baltimore, Maryland 21218	7b. ADDRESS (City, State, and ZIP Code): Bolling AFB, Washington DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR / 8b. OFFICE SYMBOL (If applicable): NM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: AFOSR-85-0097-B
8c. ADDRESS (City, State, and ZIP Code): Building 410, Bolling AFB - DC 20332-6448	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO [illegible] / PROJECT NO [illegible] / TASK NO [illegible] / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
FAULT TOLERANT PARALLEL IMPLEMENTATIONS OF ITERATIVE ALGORITHMS FOR OPTIMAL CONTROL PROBLEMS

12. PERSONAL AUTHOR(S)
Gerard G. L. Meyer and Howard L. Weinert

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 1/1/85 TO 12/31/87	1988 January 21	39

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Algorithm, parallelism, optimal control, fault tolerant.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This final report briefly describes progress on research in algorithms for optimal control problems. The principal research focus has been on a new approach to the parallel implementation of iterative algorithms for optimal control based on a two level parametrization of optimality conditions, and a secondary research focus has been the investigation of fault detection in the type of computational networks used for optimal control computations. Publications describing the results in detail are listed.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION: Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL: Major James M. Crowley	22b. TELEPHONE (Include Area Code): (202)-767-5025 / 22c. OFFICE SYMBOL: NM

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

89 11 27 030

AFOSR-TR- 89-1387

FAULT TOLERANT PARALLEL IMPLEMENTATIONS OF ITERATIVE ALGORITHMS FOR OPTIMAL CONTROL PROBLEMS

Gerard G.L. Meyer and Howard L. Weinert

Electrical and Computer Engineering Department

The Johns Hopkins University

Baltimore, Maryland 21218

FINAL SCIENTIFIC REPORT

under

Contract Number AFOSR-85-0097-B

for the period

January 1, 1985 to December 31, 1987

TABLE OF CONTENTS

ABSTRACT

This annual report briefly describes progress on research in algorithms for optimal control problems. The principal research focus has been on a new approach to the parallel implementation of iterative algorithms for optimal control based on a two level parametrization of optimality conditions, and a secondary research focus has been the investigation of fault detection in the type of computational networks used for optimal control computations. Publications describing the results in detail are listed.

1. RESEARCH OBJECTIVES AND STATUS

The principal focus of our research is a new systematic approach to design optimal control algorithms that may be implemented on parallel machines. This approach is based on a two-level parametrization of first-order optimality conditions. The first level of parametrization is concerned with the decrease of the overall amount of operations, and the second level is concerned with parallelism. By introducing parametrization matrices in the first level and then factoring those matrices to exhibit the amount of parallelism desired in the second level as a function of the number of processing elements to be used, the resulting optimality conditions may be tailored to the computing network on which the computations are to be performed. The results have been published in the Journal of Parallel and Distributed Computing [1], and have been presented at the 1987 Annual International Conference on Parallel Processing [5], the 1987 Allerton Conference on Communication, Control and Computing [6], the Third SIAM Conference on Parallel Processing for Scientific Computing [7], and are also the subject of L. J. Podrazik's Ph. D. dissertation [8]. The research results concerning the convergence properties of relaxation algorithms that are used in parallel schemes have been published in Mathematical Programming [2].

The second research focus has been the investigation of fault detection in computational networks of the type analyzed in the course of our investigation of parallelism for optimal control. We have concentrated our effort in the study of system level fault models,and the results have been accepted for publication in the IEEE Transactions on Computers [4], and are also the subject of M. A. Kennedy's Ph. D. dissertation [3].

2. PH.D. COMPLETED: M.A. KENNEDY, MAY 1987

A STRUCTURAL APPROACH TO A SYSTEM LEVEL FAULT MODEL

ABSTRACT

The widespread use of computers, both large and small, has lead to an increase in the fault problem. This problem is most acute while the system is operating, because testing and fault diagnosis may not be possible during operation. One method of addressing this problem is to use the processing power of the system itself to enhance its ability to diagnose faults. System level fault models provide a framework for addressing this problem. These models represent a system in terms of its constituent processing elements, its faults, the tests to identify the faults, and the relationship between the faults and the test outcomes. This work considers the system level fault model of Preparata, Metze, and Chien which envisions a multiple computer system as a collection of processing elements and test links. The focus of the work is the relationship between the test link structure and the system diagnosis properties. Results include a test-link based method for partitioning the processing elements that provides both a new measure for comparing systems and an indication of the complexity of identifying the maximum diagnosability number of a system. This partitioning concept leads to new diagnosability conditions that fill in the gap between existing diagnosability conditions and their relationship to properties of the test link structure. The partition is also used to synthesize improved algorithms for identifying the maximum diagnosability number of a system. Turning to implied faulty set properties useful in diagnosis, results for both constrained and unconstrained system structures are presented. Finally, these properties are incorporated into diagnosis algorithms.

3. Ph.D. COMPLETED: L. J. PODRAZIK, DECEMBER 1987

PARALLEL IMPLEMENTATIONS OF GRADIENT BASED ITERATIVE ALGORITHMS FOR OPTIMAL CONTROL PROBLEMS

ABSTRACT

The primary objective of this research is to develop new parallel techniques for solving optimal control problems that occur in online real-time applications. In view of the availability of inexpensive yet powerful hardware, the use of parallel processing techniques is proposed to satisfy both the speed constraints imposed by a real-time setting as well as the reliability requirements of an online system. Unlike previous parallel approaches to the solution of optimal control problems, the goal is to obtain an efficient solution by structuring the control algorithms to exhibit parallelism which match the given machine architecture. In order to achieve the goal, this work reexamines optimal control problems from the perspective of their first-order optimality conditions so that the issues of parallelism and machine architecture may be considered in the forefront of the algorithm synthesis. Results include the development of an efficient parallel procedure for gradient evaluation. Embedded in the parallel gradient evaluation procedure is a new technique for solving first-order linear recurrence systems which is synthesized as a function of the number of available computers. The synthesis approach for parallel recurrence solvers is also new and uses matrix factorization techniques to organize the computations for the given parallel environment. The results for parallel gradient evaluation are then exploited to produce efficient parallel implementations of iterative gradient based techniques to solve the linear quadratic regulator optimal control problem with hard control bounds. Finally, a practical multi-computer architecture is presented to provide an integrated parallel environment for the solution of time critical optimal control problems.

4. PUBLICATIONS: JANUARY 1987 - DECEMBER 1987

[1] G G. L. Meyer and L. J. Podrazik, A Parallel First-Order Linear Recurrence Solver, *J. Parallel and Distributed Computing*, vol. 4, 1987, pp. 117-132.

[2] G. G. L. Meyer, Convergence of Relaxation Algorithms by Averaging, *Mathematical Programming*, vol. 40, no. 1, 1988.

[3] M. A. Kennedy, A Structural Approach to a System Level Fault Model, a dissertation submitted the Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy, May 1987.

[4] G. G. L. Meyer and M. A. Kennedy, The PMC System Level Fault Model: Cardinality Properties of the Implied Faulty Sets, *IEEE Trans. on Computers*, accepted for publication August 1987.

[5] G. G. L. Meyer and L. J. Podrazik, Parallel Implementations of Gradient Based Iterative Algorithms for a Class of Discrete Optimal Control Problems, *Proc. 1987 International Conference on Parallel Processing*, St. Charles, Illinois, August 17-21, 1987.

[6] G. G. L. Meyer and L. J. Podrazik, Parallel Gradient Projection Algorithms to Solve the Discrete LQR Optimal Control Problem with Hard Control Bounds, *Proc. Twenty-Fifth Annual Allerton Conference on Communication, Control and Computing*, Allerton House, Monticello, Illinois, September 30-October 2, 1987.

[7] G. G. L. Meyer and L. J. Podrazik, Parallel Iterative Algorithms to Solve the Discrete LQR Optimal Control Problem with Hard Control Bounds, *Third SIAM Conference on Parallel Processing for Scientific Computing*, Los

Angeles, California, December 1-4, 1987.

[8] L. J. Podrazik, Parallel Implementations of Gradient Based Iterative Algorithms for Optimal Control Problems, a dissertation submitted the Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy, December 1987.

5. PERSONNEL

Principal Investigators: G. G. L. Meyer and H. L. Weinert

Research Assistants (Ph.D. Graduate Students)

A. Balogh

A. Harrington

M. A. Kennedy

L. J. Podrazik

V. K. Marianov

6. APPENDIX I

JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 4, 117-132 (1987)

A Parallel First-Order Linear Recurrence Solver*

GERARD G. L. MEYER AND LOUIS J. PODRAZIK

Electrical Engineering and Computer Science Department, The Johns Hopkins University, Baltimore, Maryland 21218

Received March 17, 1986

In this paper we present a parallel procedure for the solution of first-order linear recurrence systems of size N when the number of processors p is small in relation to N. We show that when $1 < p^2 \leq N$, a first-order linear recurrence system of size N can be solved in $5(N-1)/(p+1)$ steps on a p processor SIMD machine and at most $5(N-\frac{1}{2})/(p+\frac{3}{2})$ steps on a p processor MIMD machine. As a special case, we further show that our approach precisely achieves the lower bound $2(N-1)/(p+1)$ for solving the parallel prefix problem on a p processor machine. © 1987 Academic Press, Inc.

I. INTRODUCTION

In this paper we present a parallel algorithm to solve the well-known first-order linear recurrence system $R\langle N, 1\rangle$ when the number of processors p is small in relation to N, and where $R\langle N, 1\rangle$ is defined as follows:

$R\langle N, 1\rangle$: Given N, given $b = (b_1, b_2, \ldots, b_N)$,
and given $a = (a_2, a_3, \ldots, a_N)$,
compute $x = (x_1, x_2, \ldots, x_N)$
such that $x_1 = b_1$ and

$$x_i = a_i x_{i-1} + b_i \text{ for } i = 2, 3, \ldots, N.$$

We present both SIMD and MIMD versions of the algorithm. We analyze the SIMD version by first considering a simplified shared memory model of parallel computation that facilitates comparison with previous work. In that

* This work was supported by the Air Force Office of Scientific Research under Contract AFOSR-85-0097.

model the parallelism exhibited by the proposed algorithm is examined in terms of data dependencies only, therefore allowing us to determine the idealized performance of the procedure. We then consider a second model of computation which consists of a SIMD p processor ring configuration with a broadcasting capability. In that model interprocessor communication is taken into account, and a more realistic analysis of the algorithm is performed. The MIMD version of the algorithm is analyzed by considering the same simplified model as in the SIMD case, with the exception that the same operation need not be performed by all processors at the same time. Finally, we observe that the algorithm can be mapped efficiently to a MIMD p processor ring configuration with a broadcasting capability.

Many algorithms have been proposed to solve linear recurrences in parallel, each with different objectives. Earlier results assumed the availability of an unlimited number of processing elements and were concerned with determining the number of processors necessary to achieve minimal computational time [3, 6, 8, 10]. Later, limited processor solutions were considered. Chen *et al.* [4] presented a SIMD algorithm that solved Mth-order systems in $(N/p)(2M^2 + 3M) + O[M^2\log_2(p/M)]$ steps, but did not discuss any specific parallel implementation. Gajski [5] improved upon this result by performing the SIMD computation in less than $(N/p)(2M^2 + 3M)$ steps using $p \leqslant N^{1/2}$ processors in a shared memory architecture. In this paper we show that by using a SIMD p processor ring network modified to support global broadcasting, the number of steps required to solve a first-order linear recurrence of size $N \geqslant p^2$ is $5(N - 1)/(p + 1)$. This improves upon the results of [4, 5] for the first-order case when $N > p^2$. Our approach is a generalization of the matrix factorization technique presented in [9], and it reduces to the SIMD procedure presented in [5] when $N = p^2$.

Moreover, when $a_i = 1$, for all i in $[1, 2, \ldots, N]$, $R\langle N, 1\rangle$ reduces to a particular form of the parallel prefix problem. For $N > p$, Kruskal *et al.* [7] present an algorithm which solves the parallel prefix problem in $2N/p + 2\log_2 p - 2$ steps. Snir [11] improves upon this approach when $N \geqslant p^2$ by solving the problem in $2N/(p + 1) + O(1)$ steps, which is within a constant additive term of the lower bound. In the case of parallel prefix, we show that our algorithm precisely achieves the lower bound $2(N - 1)/(p + 1)$ established by Snir [11] when $N \geqslant p^2$.

Carlson and Sugla [2] considered mapping the computation of first-order linear recurrence systems to perfect shuffle and cube-connected systems. A common feature of Carlson's algorithm and our algorithm is that the computation is organized so that the transfer of input and output data can be performed concurrently with the execution of the algorithm, thus providing a balance between I/O and processing loads.

An important problem related to the parallel solution of $R\langle N, 1\rangle$ is the parallel evaluation of general arithmetic expressions [1]. In the case of first-order linear recurrences of size N, the problem is to efficiently evaluate x_N in parallel using p processors. In that case, we observe that when applied to computing x_N only, a straightforward application of our approach does not improve upon existing results [1, 4].

In order to specify the parallelism exhibited by our algorithms, we augment those statements which can be executed in parallel. We use the syntax

```
FORALL $i \in S$ DO IN PARALLEL
        BODY                              /* Comments */
END FORALL,
```

which indicates that the BODY may be executed concurrently for each i in the set S.

The SIMD procedure to solve $R\langle N, 1\rangle$ is presented in Section II; in Section III we discuss a parallel implementation of the algorithm, and in Section IV we present a MIMD version of our algorithm to solve $R\langle N, 1\rangle$. Finally, Section V comprises our conclusions.

II. The SIMD Algorithm

The abstract model of SIMD parallel computation (Fig. 1) considered in this section consists of a global parallel memory, p parallel processors, and an interconnection network, where all processors perform the same operation at each time step. We further simplify the model by making the following assumptions:

A1. Each arithmetic operation (addition or multiplication) is performed in unit time, referred to as a step.

A2. There are no accessing conflicts in global memory; furthermore, all data are assumed to reside in global memory initially.

A3. There is no time required to access global memory.

This simplistic model allows the parallelism of the proposed algorithm to be analyzed without introducing the added complexity of the implementation. In Section III we map the algorithm to a specific computational network and we analyze the corresponding implementation.

Given N and p, our approach to solving $R\langle N, 1\rangle$ consists of partitioning $R\langle N, 1\rangle$ into a sequence of $\lceil (N-1)/(p^2-1)\rceil$ reduced recurrence systems $R\langle n, 1\rangle$, each of size $n = p^2$, except for the last recurrence system, which may be of size less than p^2. Each $R\langle n, 1\rangle$ is then solved in parallel with its initial value taken as the final value obtained from the previously solved reduced

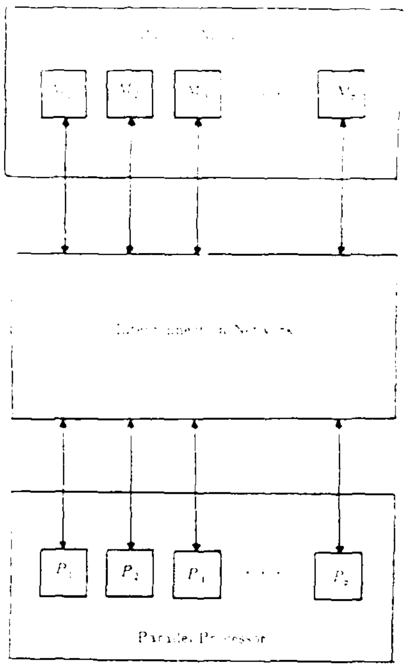


FIG. 1. The abstract parallel computational model.

recurrence system, except for the first recurrence that uses $x_1 = b_1$. The first $R\langle n, 1\rangle$ is solved as follows: (i) Each of the p processors concurrently computes a partial solution for a different x_i; (ii) after p parallel iterations p^2 partial solutions have been computed, one for each x_i, i in $[1, 2, \ldots, p^2]$, where the partial solutions for x_i, i in $[1, 2, \ldots, p]$, are precisely the solutions for $R\langle N, 1\rangle$; (iii) based upon x_p the next p partial solutions x_i, i in $[p + 1, p + 2, \ldots, 2p]$, are then updated in parallel to their correct values. After $p - 1$ parallel update iterations $R\langle n, 1\rangle$ is solved. The next reduced recurrence system of size p^2 is then solved with x_{p^2} as its initial value. We continue in this manner until the last $R\langle n, 1\rangle$ is solved. Since the initial and final values of each $R\langle n, 1\rangle$ overlap, the complete solution of $R\langle N, 1\rangle$ requires solving $\lceil (N - 1)/(p^2 - 1)\rceil$ reduced recurrence systems.

We now describe the SIMD algorithm to solve $R\langle N, 1\rangle$ when $(N - 1)/(p^2 - 1)$ is an integer. For ω in $[0, 1, \ldots, (N - 1)/(p^2 - 1)-1]$, let the index sets S_ω and T_ω be defined as

$$S_\omega = \{1 + mp + \omega(p^2 - 1) : m = 1, 2, \ldots, p - 1\}$$

and

$$T_\omega = \{1 + mp + \omega(p^2 - 1) : m = 0, 1, \ldots, p - 1\}.$$

```
 1. PROCEDURE $R(N, p, a, b)$
 2.   $x_1 := b_1$
 3.   FOR $\omega := 0$ TO $(N - 1)/(p^2 - 1) - 1$ DO              /* Solve each $R\langle n, 1\rangle$ */
 4.     FORALL $i \in S_\omega$ DO IN PARALLEL              /* Begin Coefficient Computation Phase */
 5.       $A[i, i] := a_i$;
 6.     END FORALL
 7.     FOR $i := 1$ TO $(p - 1)$ DO
 8.       FORALL $j \in S_\omega$ DO IN PARALLEL
 9.         $A[i + j, j] := a_{i+j} A[i + j - 1, j]$;
10.       END FORALL
11.     END FOR                                          /* End Coefficient Computation Phase */
12.     FORALL $i \in S_\omega$ DO IN PARALLEL                    /* Begin Partial Solution Phase */
13.       $x_i := b_i$;
14.     END FORALL
15.     FOR $i := 1$ TO $(p - 1)$ DO
16.       FORALL $j \in T_\omega$ DO IN PARALLEL
17.         $x_{i+j} := a_{i+j} x_{i+j-1} + b_{i+j}$;
18.       END FORALL
19.     END FOR                                              /* End Partial Solution Phase */
20.     FOR $i \in S_\omega$ DO                                 /* Begin Solution Update Phase */
21.       FORALL $j := 0$ TO $(p - 1)$ DO IN PARALLEL
22.         $x_{i+j} := A[i + j, i] x_{i-1} + x_{i+j}$;
23.       END FORALL
24.     END FOR                                                /* End Solution Update Phase */
25.   END FOR
26. END PROCEDURE
```

The preceding algorithm sequentially solves $(N - 1)/(p^2 - 1)$ reduced recurrence systems of size p^2, each in parallel. Each reduced system is solved in three phases: the coefficient computation phase consists of the execution of loops 4 and 7 and computes all coefficients of the form $a_{i+j}a_{i+j-1} \cdots a_i$ which are needed later during the solution updates; the partial solution phase consists of the execution of loops 12 and 15 and computes p^2 partial solutions, in which the first p partial solutions are the actual solutions; and finally, the solution update phase consists of the execution of loop 20 in which the coefficients computed in the first phase are used to update the next p estimates at each iteration. The complete solution to $R\langle N, 1\rangle$ is therefore obtained after executing $(N - 1)/(p^2 - 1)$ iterations of loop 3. An example illustrating the computations performed by the algorithm is given in Fig. 2 for the case $N = 17$ and $p = 3$, where the notation $\underline{x}_i$ is used to indicate a correct value for the solution. Note that each computational level may be performed in parallel using at most three processors.

Our model assumptions imply that we need only consider computational operations when determining the number of steps required by the algorithm.

Computational phase	Time step	Parallel computations		
Coefficient computation phase		—	$A[4,4] = a_4$	$A[7,7] = a_7$
	1	—	$A[5,4] = a_5A[4,4]$	$A[8,7] = a_8A[7,7]$
	2	—	$A[6,4] = a_6A[5,4]$	$A[9,7] = a_9A[8,7]$
Partial solution phase		$\underline{x}_1 = b_1$	$x_4 = b_4$	$x_7 = b_7$
	3, 4	$\underline{x}_2 = a_2\underline{x}_1 + b_2$	$x_5 = a_5x_4 + b_5$	$x_8 = a_8x_7 + b_8$
	5, 6	$\underline{x}_3 = a_3\underline{x}_2 + b_3$	$x_6 = a_6x_5 + b_6$	$x_9 = a_9x_8 + b_9$
Solution update phase	7, 8	$\underline{x}_4 = A[4,4]\underline{x}_3 + x_4$	$\underline{x}_5 = A[5,4]\underline{x}_3 + x_5$	$\underline{x}_6 = A[6,4]\underline{x}_3 + x_6$
	9, 10	$\underline{x}_7 = A[7,7]\underline{x}_6 + x_7$	$\underline{x}_8 = A[8,7]\underline{x}_6 + x_8$	$\underline{x}_9 = A[9,7]\underline{x}_6 + x_9$
Coefficient computation phase		—	$A[12,12] = a_{12}$	$A[15,15] = a_{15}$
	11	—	$A[13,12] = a_{13}A[12,12]$	$A[16,15] = a_{16}A[15,15]$
	12	—	$A[14,12] = a_{14}A[13,12]$	$A[17,15] = a_{17}A[16,15]$
Partial solution phase		$\underline{x}_9 = \underline{x}_9$	$x_{12} = b_{12}$	$x_{15} = b_{15}$
	13, 14	$\underline{x}_{10} = a_{10}\underline{x}_9 + b_{10}$	$x_{13} = a_{13}x_{12} + b_{13}$	$x_{16} = a_{16}x_{15} + b_{16}$
	15, 16	$\underline{x}_{11} = a_{11}\underline{x}_{10} + b_{11}$	$x_{14} = a_{14}x_{13} + b_{14}$	$x_{17} = a_{17}x_{16} + b_{17}$
Solution update phase	17, 18	$\underline{x}_{12} = A[12,12]\underline{x}_{11} + x_{12}$	$\underline{x}_{13} = A[13,12]\underline{x}_{11} + x_{13}$	$\underline{x}_{14} = A[14,12]\underline{x}_{11} + x_{14}$
	19, 20	$\underline{x}_{15} = A[15,15]\underline{x}_{14} + x_{15}$	$\underline{x}_{16} = A[16,15]\underline{x}_{14} + x_{16}$	$\underline{x}_{17} = A[17,15]\underline{x}_{14} + x_{17}$

FIG. 2. Solution of $R\langle 17, 1\rangle$ for three-processor SIMD computer.

Therefore we must examine those computations performed in loops 7, 15, and 20. The execution of loop 7 is performed $p - 1$ times, each iteration requiring $p - 1$ processors to perform a single multiplication concurrently; thus loop 7 requires $p - 1$ steps. Both loops 15 and 20 are iterated $p - 1$ times, each iteration requiring p processors to concurrently perform a multiplication followed by an addition. Thus, loops 15 and 20 each require $2(p - 1)$ steps. The total number of steps required to solve each reduced recurrence system $R\langle p^2, 1\rangle$ using p processors is therefore $5(p - 1)$, and hence the resulting theorem follows.

THEOREM 1. *Given N and p such that $(N - 1)/(p^2 - 1)$ is an integer, the number of steps required to solve the linear recurrence system $R\langle N, 1\rangle$ using a SIMD parallel computer with p processors is $5(N - 1)/(p + 1)$.*

If $(N - 1)/(p^2 - 1)$ is not an integer, our approach to solving $R\langle N, 1\rangle$ requires that we solve the reduced recurrence system $R\langle n_r, 1\rangle$, where $n_r < p^2$. One approach to solving $R\langle n_r, 1\rangle$ is to use a technique which is known to be efficient whenever $n_r < p^2$. Applicable techniques include the algorithms presented by Chen *et al.* [4] and Kogge and Stone [6]; however, these techniques are not desirable because they require the machine to store and execute multiple algorithms based upon the size of the recurrence system. A less efficient but more practical approach to solving $R\langle n_r, 1\rangle$ consists of using the proposed technique to solve the augmented system $R\langle p^2, 1\rangle$ and simply terminate the computation when the last x_i is computed. In that case the number of steps required by the algorithm is at most $\lceil (N - 1)/(p^2 - 1)\rceil 5(p - 1)$.

Finally, we make the observation that the above SIMD algorithm most notably differs from the approaches presented in [4, 5] in that our approach partitions the problem and solves a series of reduced recurrences of size p^2 sequentially. However, when $N = p^2$, our approach reduces to that of [5], except that Gajski presents the coefficient computation and partial solution phases as a single computational phase. Moreover, when $N \geqslant p^2$, the algorithm of Chen *et al.* [4] is less efficient than Gajski's as a result of implementing an extra computational phase in which a separate first-order recurrence of size p is solved using p processors, requiring an additional $2 \log_2 p$ steps.

When N and p are powers of two, the algorithm of Chen *et al.* requires $5N/p + 2 \log_2 p - 5$ steps [4] and when N/p^2 is an integer, Gajski's SIMD algorithm requires $(N/p^2)(5p - 3) - 2$ steps [5], whereas when $(N - 1)/(p^2 - 1)$ is an integer, our SIMD algorithm requires only $5(N - 1)/(p + 1)$ steps. For example, when $N = 2^{18}$ and $p = 2^3$, the numbers of steps required by the SIMD algorithms presented in [4, 5] and this paper are 163,841, 151,550, and 145,635, respectively.

Finally, when $a_i = 1$, for all i in $[1, 2, \ldots, N]$, $R\langle N, 1\rangle$ is a particular form of the parallel prefix problem and reduces to computing the cascade sums

$(b_1 + b_2), (b_1 + b_2 + b_3), \ldots, (b_1 + b_2 + \cdots + b_N)$ in parallel using p processors. The following corollary is a direct consequence of Theorem 1.

COROLLARY 1. *Given N and p such that $(N - 1)/(p^2 - 1)$ is an integer the number of steps required to solve the parallel prefix problem using a SIMD parallel computer with p processors is $2(N - 1)/(p + 1)$.*

Thus, when $(N - 1)/(p^2 - 1)$ is an integer, our SIMD algorithm precisely achieves the parallel prefix computational lower bound $2(N - 1)/(p + 1)$ established by Snir [11]. This result improves upon existing approaches to solving the parallel prefix problem when $N > p^2$. In that case the parallel prefix problem is solved in $2N/(p + 1) + O(1)$ steps by Snir's algorithm [11], $2(N/p) + \log_2 p - 2$ steps by the data-independent algorithm presented by Kruskal *et al.* [7], and $(N/p^2)(2p - 1) - 1$ steps by Gajski's algorithm [5].

III. THE SIMD PARALLEL IMPLEMENTATION

The abstract model of SIMD parallel computation presented in Section II neglected the issues of data organization and alignment as well as communication overhead, all of which are highly machine dependent. We now present a parallel implementation of the proposed algorithm that takes these issues into account. The SIMD model of computation considered in this section (Fig. 3) consists of p processors executing the same operation in lock step, with each processor containing its own local storage. The processors are interconnected by a unidirectional ring network in which processor i transfers

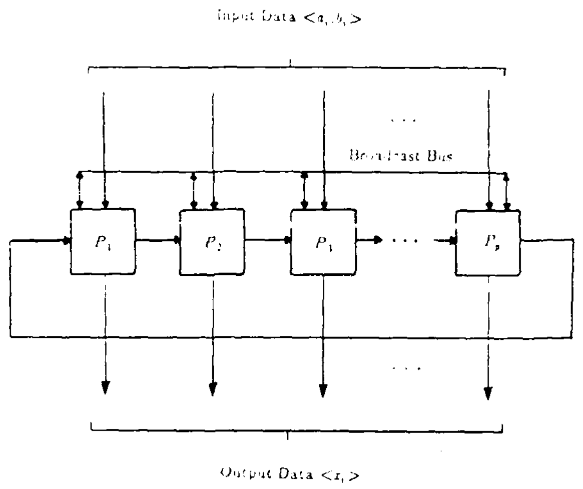


FIG. 3. The practical parallel computational model.

data to processor $i + 1$, i in $[1, 2, \ldots, p - 1]$, and processor p transfers data to processor 1. Furthermore, we assume that the network possesses a broadcasting capability that allows any processor to broadcast data to all other processors. The time required by the algorithm will be determined under the following assumptions:

A1. Each arithmetic operation (addition or multiplication) is performed in unit time, referred to as a step.

A2. Interprocessor transfers require one step.

A3. Data broadcasts require one step.

A4. Each a_i and b_i required by a processor is assumed to reside in the local memory of that processor initially.

In order to determine an efficient processor assignment, we first make the observation that the p consecutive partial solutions updated at each iteration of the update phase of the algorithm must reside in a different processor. Furthermore, both the coefficient computation phase and the partial solution phase of the algorithm exhibit explicit data dependencies which must be preserved. These constraints can be satisfied if we rotate the processor assignment at each parallel iteration of the algorithm, and in that case, the algorithm can be directly mapped to a SIMD p processor unidirectional ring network with broadcasting capability. Figure 4 illustrates such a processor assignment and the corresponding communication requirements for the case $N = 17$ and $p = 3$.

We now present the algorithm to solve $R\langle N, 1\rangle$ as executed by processor k, for all k in $[1, 2, \ldots, p]$.

```
 1. PROCEDURE $R_k(N, p, a, b)$
 2.   $x_1 := b_1$
 3.   FOR $\omega := 0$ TO $(N - 1)/(p^2 - 1) - 1$ DO            /* Solve each $R\langle n, 1\rangle$ */
 4.     $A[i, i] := a_i$;                       /* Begin Coefficient Computation Phase */
          /* $i = 1 + (k - 1)p + \omega(p^2 - 1)$ */
 5.     FOR $i := 1$ TO $(p - 1)$ DO
 6.       $A[i + j, j] := a_{i+j} A[i + j - 1, j]$;
          /* $j = (1 + (k - i - 1)p) \bmod p^2 + \omega(p^2 - 1)$ */
 7.     END FOR                                 /* End Coefficient Computation Phase */
 8.     IF $k = 1$ THEN $x_i := x_i$ ELSE $x_i := b_i$;      /* Begin Partial Solution Phase */
          /* $i = 1 + (k - 1)p + \omega(p^2 - 1)$ */
 9.     FOR $i := 1$ TO $(p - 1)$ DO
10.       $x_{i+j} := a_{i+j} x_{i+j-1} + b_{i+j}$;
          /* $j = (1 + (k - i - 1)p) \bmod p^2 + \omega(p^2 - 1)$ */
11.     END FOR                                 /* End Partial Solution Phase */
12.     FOR $m := 1$ TO $(p - 1)$ DO                  /* Begin Solution Update Phase */
13.       $x_{i+j} := A[i + j, i] x_{i-1} + x_{i+j}$;
          /* $i = 1 + mp + \omega(p^2 - 1)$, $j = (p - m + k - 1) \bmod p$ */
14.     END FOR                                 /* End Solution Update Phase */
15.   END FOR
16. END PROCEDURE
```

Computational phase	Time step	Parallel computations: Processor 1	Parallel computations: Processor 2	Parallel computations: Processor 3
Processor resident data		$a_i, i = 6, 8, 14, 16$	$a_i, i = 2, 4, 9, 10, 12, 17$	$a_i, i = 3, 5, 7, 11, 13, 15$
		$b_i, i = 1, 6, 8, 14, 16$	$b_i, i = 2, 4, 9, 10, 12, 17$	$b_i, i = 3, 5, 7, 11, 13, 15$
Coefficient computation phase	1	—	$A[4, 4] = a_4$ →	$A[7, 7] = a_7$ →
	2, 3	$A[8, 7] = a_8A[7, 7]$ →	—	$A[5, 4] = a_5A[4, 4]$ →
	4	$A[6, 4] = a_6A[5, 4]$	$A[9, 7] = a_9A[8, 7]$	—
Partial solution phase	5	$\underline{x}_1 = b_1$ →	$x_4 = b_4$ →	$x_7 = b_7$ →
	6, 7, 8	$x_8 = a_8x_7 + b_8$ →	$\underline{x}_2 = a_2\underline{x}_1 + b_2$ →	$x_5 = a_5x_4 + b_5$
	9, 10, 11	$x_6 = a_6x_5 + b_6$	$x_9 = a_9x_8 + b_9$	$x_3 = a_3\underline{x}_2 + b_3$ ⁝
Solution update phase	12, 13, 14	$\underline{x}_6 = A[6, 4]\underline{x}_3 + x_6$ ⁝	$\underline{x}_4 = A[4, 4]\underline{x}_3 + x_4$	$\underline{x}_5 = A[5, 4]x_3 + x_5$
	15, 16	$\underline{x}_8 = A[8, 7]\underline{x}_6 + x_8$	$\underline{x}_9 = A[9, 7]\underline{x}_6 + x_9$	$\underline{x}_7 = A[7, 7]x_6 + x_7$
Coefficient computation phase	17	—	$A[12, 12] = a_{12}$ →	$A[15, 15] = a_{15}$ →
	18, 19	$A[16, 15] = a_{16}A[15, 15]$ →	$\underline{x}_9$ →	$A[13, 12] = a_{13}A[12, 12]$ →
	20	$A[14, 12] = a_{14}A[13, 12]$	$A[17, 15] = a_{17}A[16, 15]$	$\underline{x}_9$ →
Partial solution phase	21	$\underline{x}_9$ →	$x_{12} = b_{12}$ →	$x_{15} = b_{15}$ →
	22, 23, 24	$x_{16} = a_{16}x_{15} + b_{16}$ →	$\underline{x}_{10} = a_{10}\underline{x}_9 + b_{10}$ →	$x_{13} = a_{13}x_{12} + b_{13}$ →
	25, 26, 27	$x_{14} = a_{14}x_{13} + b_{14}$	$x_{17} = a_{17}x_{16} + b_{17}$	$x_{11} = a_{11}x_{10} + b_{11}$ ⁝
Solution update phase	28, 29, 30	$x_{14} = A[14, 12]\underline{x}_{11} + x_{14}$ ⁝	$\underline{x}_{12} = A[12, 12]x_{11} + x_{12}$	$x_{13} = A[13, 12]x_{11} + x_{13}$
	31, 32	$\underline{x}_{16} = A[16, 15]\underline{x}_{14} + x_{16}$	$\underline{x}_{17} = A[17, 15]x_{14} + x_{17}$	$\underline{x}_{15} = A[15, 15]x_{14} + x_{15}$
Processor resident results		$x_i, i = 1, 6, 8, 14, 16$	$x_i, i = 2, 4, 9, 10, 12, 17$	$x_i, i = 3, 5, 7, 11, 13, 15$

FIG. 4. Processor assignment for SIMD solution of $R\langle 17, 1\rangle$ with $p = 3$. Notation: → indicates that the data evaluated at the current step are transferred to the processor on the right; ⁝ indicates that the data evaluated at the current step are broadcast to all processors.

Our model assumptions imply that we must consider interprocessor communication in addition to operational count when determining the number of steps required by the algorithm. Therefore, we must examine the computations and interprocessor transfers performed in loops 5, 9, and 12. Each iteration of loop 5 requires an interprocessor transfer of $A[i + j - 1, j]$ followed by a single multiplication. Thus, loop 5 requires $2(p - 1)$ steps. Loop 9 is iterated $p - 1$ times, each iteration requiring an interprocessor transfer of x_{i+j-1} followed by a multiplication and an addition. Thus, loop 9 requires $3(p - 1)$ steps. Loop 12 is also iterated $p - 1$ times, each iteration requiring a data broadcast of x_{i-1} followed by a multiplication and an addition. Thus, loop 12 also requires $3(p - 1)$ steps. The total number of steps required to solve each reduced recurrence $R\langle p^2, 1\rangle$ using p processors is therefore $8(p - 1)$, and hence, the resulting theorem follows.

THEOREM 2. *Given N and p such that $(N - 1)/(p^2 - 1)$ is an integer, the number of steps required to solve a linear recurrence system $R\langle N, 1\rangle$ using a SIMD parallel computer with p processors is $8(N - 1)/(p + 1)$.*

Among the existing SIMD algorithms to solve $R\langle N, 1\rangle$, the SIMD algorithm presented by Gajski [5] can be most efficiently mapped to a unidirectional ring network with broadcasting capability. Based upon the assumptions made in this section, when N/p^2 is an integer the number of steps required by Gajski's approach to solve $R\langle N, 1\rangle$ is $(N/p^2)(8p - 5) - 3$, and therefore when $N > p^2$ our approach is more efficient than Gajski's when implemented upon a ring network capable of broadcasting.

Finally, we make the observation that the algorithm does not require all of the inputs a_i and b_i in order for the processing to begin. Specifically, the algorithm requires $p^2 - 1$ a_i and p^2 b_i for every $5(p - 1)$ computational steps, corresponding to solving each $R\langle p^2, 1\rangle$ in sequence. Similarly, the outputs x_i are produced in blocks of $p^2 - 1$ at a time. This suggests that I/O could be overlapped with the computation, providing a balance between I/O and processing loads, and therefore the deletion of assumption A4 has a negligible effect if one assumes that I/O and processing can be done concurrently.

IV. THE MIMD ALGORITHM

In this section we again consider the simplistic model of computation given in Section II with the exception that we no longer require all processors to execute the same operation at each step; that is, we now consider a MIMD implementation in which we neglect the issues of data organization and alignment as well as communication overhead.

The MIMD approach for solving $R\langle N, 1\rangle$ is based upon the observation that only $p - 1$ processors are needed at each iteration of the coefficient computation phase. Assuming $(N-1)/(p^2-1)$ to be an integer, the total number of multiplications required to compute all necessary coefficients is $(N-1)(p-1)/(p+1)$, $p-1$ of which may be performed concurrently at each step. Therefore, all of the required coefficients can be computed in $(N-1)/(p+1)$ steps using $p-1$ processors. This leaves one processor free for $(N-1)/(p+1)$ steps, allowing us to expand the size of the recurrence by at most $n_0 = \lfloor (N-1)/2(p+1) \rfloor$ and use the free processor to solve the reduced system $R\langle n_0, 1\rangle$ concurrently. Thus, using a MIMD approach we can solve the entire system $R\langle N+n_0, 1\rangle$ in $5(N-1)/(p+1)$ steps.

Given a recurrence system of size N and the number of processors p the following lemma expresses n_0 in terms of N and p only.

LEMMA 1. *Given N and p, $n_0 = \lceil [N-(p+2)]/(2p+3) \rceil$.*

Given N and p, our MIMD approach to solving $R\langle N, 1\rangle$ consists of partitioning $R\langle N, 1\rangle$ into a sequence of $\lceil (N-n_0-1)/(p^2-1) \rceil + 1$ reduced recurrences. The first recurrence is of size n_0+1 and all others are of size p^2, except for the last recurrence, which may be of size less than p^2. The coefficient computation phase of the algorithm uses $p-1$ processors to compute all needed coefficients for all of the reduced systems. Concurrent with this computation, the free processor computes the solution to $R\langle n_0+1, 1\rangle$. Each subsequent $R\langle n, 1\rangle$ is then solved in the same manner as in the SIMD algorithm by executing a partial solution phase followed by a solution update phase. The complete solution is obtained after solving all $\lceil (N-n_0-1)/(p^2-1) \rceil + 1$ reduced recurrences.

We now present the MIMD algorithm to solve $R\langle N, 1\rangle$ when $(N-n_0-1)/(p^2-1)$ is an integer. As in the SIMD case, it is not difficult to modify the algorithm if the above assumption is not satisfied by simply terminating the computation at the point when the last x_i is updated. For ω in $[0, 1, \ldots, (N-n_0-1)/(p^2-1)-1]$, we now define the index sets U_ω and V_ω as

$$U_\omega = \{1+n_0+mp+\omega(p^2-1) : m = 1, 2, \ldots, p-1\}$$

and

$$V_\omega = \{1+n_0+mp+\omega(p^2-1) : m = 0, 1, \ldots, p-1\}.$$

```
1. PROCEDURE $R(N, p, a, b)$
2.   $x_1 := b_1$
3.   FOR $i := 2$ to $n_0 + 1$ DO                      /* Solve $R\langle n_0, 1\rangle$ */
4.     $x_i = a_i x_{i-1} + b_i$
5.   END FOR                                    /* End $r\langle n_0, 1\rangle$ Solution */
6.   FOR $\omega := 0$ TO $(N - n_0 - 1)/(p^2 - 1) - 1$ DO /* Begin Coefficient Computation Phase */
7.     FORALL $i \in U_\omega$ DO IN PARALLEL
8.       $A[i, i] := a_i;$
9.     END FORALL
```

```
10.      FOR $i$ := 1 TO $(p - 1)$ DO
11.        FORALL $j \in U_\omega$ DO IN PARALLEL
12.          $A[i + j, j] := a_{i+j} A[i + j - 1, j]$;
13.        END FORALL
14.      END FOR
15.    END FOR                                   /* End Coefficient Computation Phase */
16.    FOR $\omega$ := 0 TO $(N - n_0 - 1)/(p^2 - 1) - 1$ DO        /* Solve each $R\langle n, 1\rangle$ */
17.      FORALL $i \in U_\omega$ DO IN PARALLEL           /* Begin Partial Solution Phase */
18.        $x_i := b_i$;
19.      END FORALL
20.      FOR $i$ := 1 TO $(p - 1)$ DO
21.        FORALL $j \in U_\omega$ DO IN PARALLEL
22.          $x_{i+j} := a_{i+j} x_{i+j-1} + b_{i+j}$;
23.        END FORALL
24.      END FOR                                 /* End Partial Solution Phase */
25.      FOR $i \in U_\omega$ DO                       /* Begin Solution Update Phase */
26.        FORALL $j$ := 0 TO $(p - 1)$ DO IN PARALLEL
27.          $x_{i+j} := A[i + j, i] x_{i-1} + x_{i+j}$;
28.        END FORALL
29.      END FOR
30.    END FOR                                   /* End Solution Update Phase */
31.  END PROCEDURE
```

Note that (i) the coefficient computation phase of the SIMD algorithm has been modified so as to compute the necessary coefficients for all $R\langle n, 1\rangle$ before the first reduced recurrence is solved in parallel; and (ii) the processor that is idle during the SIMD coefficient computation phase is now used to concurrently compute the solution to $R\langle n_0 + 1, 1\rangle$. An example illustrating the computations performed by the MIMD algorithm is given in Fig. 5 for the case $N = 19$ and $p = 3$.

Based upon the MIMD model considered in this section, we conclude that the time required by the MIMD algorithm is determined by the computational operations performed in loops 3, 6, and 16. Loops 3 and 6 are executed concurrently, using 1 and $p - 1$ processors, respectively. Loop 6 requires $(N - n_0 - 1)/(p + 1)$ steps, and the quantity n_0 has been defined so that loop 3 requires at most the same number of steps as loop 6. All p processors are used in executing loop 16, and thus loop 16 requires $4(N - n_0 - 1)/(p + 1)$ steps. The number of steps required by the MIMD algorithm is therefore $5(N - n_0 - 1)/(p + 1)$ steps. Thus, the resulting theorem follows.

THEOREM 3. *Given N and p such that $N \geqslant p^2 + \frac{1}{2}p - 1$, the number of steps required to solve a linear recurrence system $R\langle N, 1\rangle$ using a MIMD parallel computer with p processors is at most $\lceil (N - \frac{1}{2})/(p + 3/2)(p - 1)\rceil 5(p - 1)$.*

When $N = p^2 + \frac{1}{2}p - 1$, our approach reduces to the MIMD algorithm presented in [5], in which the number of steps required to solve $R\langle N, 1\rangle$ is at most $\lceil (N - 1)/(p^2 + \frac{1}{2}p - 2)\rceil 5(p - 1)$. When $N > p^2 + \frac{1}{2}p - 1$, our MIMD

Computational phase	Time step	Parallel computations					
Coefficient computation		x_1	b_1	$A[6,6]$	a_6	$A[9,9]$	a_9
$R\langle n_0+1, 1\rangle$	1	x_2	a_2x_1	$A[7,6]$	$a_7A[6,6]$	$A[10,9]$	$a_{10}A[9,9]$
solution phase	2	x_2	$x_2 + b_2$	$A[8,6]$	$a_8A[7,6]$	$A[11,9]$	$a_{11}A[10,9]$
				$A[14,14]$	a_{14}	$A[17,17]$	a_{17}
	3	x_3	a_3x_2	$A[15,14]$	$a_{15}A[14,14]$	$A[18,17]$	$a_{18}A[17,17]$
	4	x_3	$x_3 + b_3$	$A[16,14]$	$a_{16}A[15,14]$	$A[19,17]$	$a_{19}A[18,17]$
Partial solution phase		x_3	x_3	x_6	b_6	x_9	b_9
	5, 6	x_4	$a_4x_3 + b_4$	x_7	$a_7x_6 + b_7$	x_{10}	$a_{10}x_9 + b_{10}$
	7, 8	x_5	$a_5x_4 + b_5$	x_8	$a_8x_7 + b_8$	x_{11}	$a_{11}x_{10} + b_{11}$
Solution update phase	9, 10	x_6	$A[6,6]x_5 + x_6$	x_7	$A[7,6]x_5 + x_7$	x_8	$A[8,6]x_5 + x_8$
	11, 12	x_9	$A[9,9]x_8 + x_9$	x_{10}	$A[10,9]x_8 + x_{10}$	x_{11}	$A[11,9]x_8 + x_{11}$
Partial solution phase		x_{11}	x_{11}	x_{14}	b_{14}	x_{17}	b_{17}
	13, 14	x_{12}	$a_{12}x_{11} + b_{12}$	x_{15}	$a_{15}x_{14} + b_{15}$	x_{18}	$a_{18}x_{17} + b_{18}$
	15, 16	x_{13}	$a_{13}x_{12} + b_{13}$	x_{16}	$a_{16}x_{15} + b_{16}$	x_{19}	$a_{19}x_{18} + b_{19}$
Solution update phase	17, 18	x_{14}	$A[14,14]x_{13} + x_{14}$	x_{15}	$A[15,14]x_{13} + x_{15}$	x_{16}	$A[16,14]x_{13} + x_{16}$
	19, 20	x_{17}	$A[17,17]x_{16} + x_{17}$	x_{18}	$A[18,17]x_{16} + x_{18}$	x_{19}	$A[19,17]x_{16} + x_{19}$

FIG. 5. Solution of $R\langle 19, 1\rangle$ for three-processor MIMD computer

		Parallel computations		
Computational phase	Time step	Processor 1	Processor 2	Processor 3
Processor resident data		a_i, i 8, 10, 16, 18	a_i, i 2, 3, 4, 6, 11, 12, 14, 19	a_i, i 5, 7, 9, 13, 15, 17
		b_i, i 1, 8, 10, 16, 18	b_i, i 4, 6, 11, 12, 14, 19	b_i, i 2, 3, 5, 7, 9, 13, 15, 17
Coefficient computation	1	$\underline{x}_1$ b_1 ·	$A[6, 6]$ a_6 ·	$A[9, 9]$ a_9 ·
$+ R \langle n_0 + 1, 1\rangle$	2, 3	$A[10, 9]$ $a_{10}A[9, 9]$ ·	x_2 $a_2\underline{x}_1$ ·	$A[7, 6]$ $a_7A[6, 6]$ ·
solution phase	4, 5	$A[8, 6]$ $a_8A[7, 6]$	$A[11, 9]$ $a_{11}A[10, 9]$	$\underline{x}_2$ $x_2 + b_2$ ·
	6	$\underline{x}_2$ ·	$A[14, 14]$ a_{14} ·	$A[17, 17]$ a_{17} ·
	7, 8	$A[18, 17]$ $a_{18}A[17, 17]$ ·	x_3 a_3x_2 ·	$A[15, 14]$ $a_{15}A[14, 14]$ ·
	9, 10	$A[16, 14]$ $a_{16}A[15, 14]$	$A[19, 17]$ $a_{19}A[18, 17]$	$\underline{x}_3$ $x_3 + b_3$ ·
Partial solution phase	11	$\underline{x}_3$ ·	x_6 b_6 ·	x_9 b_9 ·
	12, 13, 14	x_{10} $a_{10}x_9 + b_{10}$ ·	$\underline{x}_4$ $a_4x_3 + b_4$ ·	x_7 $a_7x_6 + b_7$ ·
	15, 16, 17	x_8 $a_8x_7 + b_8$	x_{11} $a_{11}x_{10} + b_{11}$	x_5 $a_5\underline{x}_4 + b_5$ ⁝
Solution update phase	18, 19, 20	$\underline{x}_8$ $A[8, 6]x_5 + x_8$ ⁝	$\underline{x}_6$ $A[6, 6]x_5 + x_6$	x_7 $A[7, 6]x_5 + x_7$
	21, 22	$\underline{x}_{10}$ $A[10, 9]x_8 + x_{10}$	$\underline{x}_{11}$ $A[11, 9]x_8 + x_{11}$	$\underline{x}_9$ $A[9, 9]x_8 + x_9$
Partial solution phase	23	—	x_{14} b_{14} ·	x_{17} b_{17} ·
	24, 25, 26	x_{18} $a_{18}x_{17} + b_{18}$ ·	$\underline{x}_{12}$ $a_{12}\underline{x}_{11} + b_{12}$ ·	x_{15} $a_{15}x_{14} + b_{15}$ ·
	27, 28, 29	x_{16} $a_{16}x_{15} + b_{16}$	x_{19} $a_{19}x_{18} + b_{19}$	$\underline{x}_{13}$ $a_{13}\underline{x}_{12} + b_{13}$ ⁝
Solution update phase	30, 31, 32	$\underline{x}_{16}$ $A[16, 14]x_{13} + x_{16}$ ⁝	$\underline{x}_{14}$ $A[14, 14]x_{13} + x_{14}$	$\underline{x}_{15}$ $A[15, 14]x_{13} + x_{15}$
	33, 34	$\underline{x}_{18}$ $A[18, 17]\underline{x}_{16} + x_{18}$	$\underline{x}_{19}$ $A[19, 17]x_{16} + x_{19}$	$\underline{x}_{17}$ $A[17, 17]x_{16} + x_{17}$
Processor resident results		x_i, i 1, 8, 10, 16, 18	x_i, i 4, 6, 11, 12, 14, 19	x_i, i 2, 3, 5, 7, 9, 13, 15, 17

FIG. 6 Processor assignment for MIMD solution of $R\langle 19, 1\rangle$ with p 3. Notation: · indicates that the data evaluated at the current step are transferred to the processor on the right; ⁝ indicates that the data evaluated at the current step are broadcast to all processors.

approach differs from [5] by organizing a single coefficient computation phase to compute the necessary coefficients for all $R\langle n, 1\rangle$ before the first reduced recurrence is solved in parallel, rather than including a coefficient computation phase as part of solving each reduced recurrence.

Finally, we note that, like the SIMD algorithm, the MIMD algorithm can also be mapped directly to a p processor unidirectional ring network with broadcasting capability. Figure 6 illustrates such a processor assignment and the corresponding communication requirements for the case $N = 19$ and $p = 3$.

V. Conclusions

The algorithm presented in this paper exploits the fact that for a fixed number of processors p, the parallel approach presented in [9] to solve $R\langle N, 1\rangle$ attains maximum speedup $\frac{2}{3}(p + 1)$ when $N = p^2$. When $N \geq p^2$, the structure of $R\langle N, 1\rangle$ allows the solution to be obtained by sequentially solving a series of reduced recurrences, each of size p^2, except for the last recurrence system, which may be of size less than p^2. As a result, we are able to improve upon existing approaches for solving $R\langle N, 1\rangle$ whenever $N > p^2$.

References

1. Brent, R. P. The parallel evaluation of general arithmetic expressions. *J. Assoc. Comput. Mach.* **21**, 2 (Apr. 1974), 201–206.
2. Carlson, D. A., and Sugla, B. Time and processor efficient parallel algorithms for recurrence equations and related problems. *Proc. 1984 International Conference on Parallel Processing* Aug. 21–24, 1984, pp. 310–314.
3. Chen, S. C., and Kuck, D. J. Time and parallel processor bounds for linear recurrence systems. *IEEE Trans. Comput.* **C-24**, 7 (July 1975), 701–717.
4. Chen, S. C., Kuck, D. J., and Sameh, A. H. Practical parallel band triangular system solvers. *ACM Trans. Math. Software* **4**, 3 (Sept. 1978), 270–277.
5. Gajski, D. J. An algorithm for solving linear recurrence systems on parallel and pipelined machines. *IEEE Trans. Comput.* **C-30**, 3 (Mar. 1981), 190–206.
6. Kogge, P. M., and Stone, H. S. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Comput.* **C-22**, 8 (Aug. 1973), 786–793.
7. Kruskal, C. P., Rudolph, L., and Snir, M. The power of parallel prefix. *IEEE Trans. Comput.* **C-34**, 10 (Oct. 1985), 965–968.
8. Kuck, D. J. Parallel processing of ordinary programs. In *Advances in Computers*, Vol. 15, Academic Press, New York, 1976, pp. 119–179.
9. Meyer, G. G. L., and Podrazik, L. J. A matrix factorization approach to the parallel solution of first-order linear recurrences. *Proc. 23rd Annual Allerton Conference on Communication, Control and Computing*, Oct. 2–4, 1985, pp. 243–250.
10. Sameh, A. H., and Brent, R. P. Solving triangular systems on a parallel computer. *SIAM J. Numer. Anal.* **14**, 6 (Dec. 1977), 1101–1113.
11. Snir, M. Depth-size trade-offs for parallel prefix computation. *J. Algorithms* 7, 2 (June 1986), 185–201.

7. APPENDIX II

PARALLEL IMPLEMENTATIONS OF GRADIENT BASED ITERATIVE ALGORITHMS FOR A CLASS OF DISCRETE OPTIMAL CONTROL PROBLEMS

Gerard G. L. Meyer
Electrical and Computer Engineering Department
The Johns Hopkins University
Baltimore, Maryland 21218

Louis J. Podrazik
Bendix Environmental Systems Division
Allied-Signal Inc.
Baltimore, Maryland 21284-9840

ABSTRACT

In this paper we present the parallel implementations of two iterative gradient based algorithms to solve the unconstrained linear quadratic regulator optimal control problem. We show that parallel evaluation of the step length and gradient of the quadratic cost function can be efficiently performed as a function of the number of processors. We then embed our parallel step length and gradient procedures to produce parallel implementations of the gradient and conjugate gradient methods that may be executed on an SIMD machine.

I. INTRODUCTION

Previous parallel approaches to the solution of optimal control problems [4], [6], [9], have been devised without explicitly taking into consideration the computational environment. In particular, when the number of available processors is small in relation to the problem size, the above techniques simply fold the computations to fit the number of processors. More efficient parallel algorithms may be devised by considering the computational environment throughout the algorithm synthesis. Toward that end, we present in this paper a parallel procedure for gradient evaluation which is formulated as a function of the number of available processors. Although presented in the context of unconstrained optimal control, our results for gradient computation are also applicable to constrained problems. Furthermore, we show that the step size obtained as a result of the line search performed at each iteration may also be efficiently computed in parallel. We then combine the techniques for parallel gradient evaluation and step size determination to produce parallel implementations of the best-step steepest descent method and the Fletcher-Reeves conjugate gradient method to solve the linear quadratic regulator control problem [5].

It is well known that a closed loop feedback form solution exists for the linear quadratic regulator control problem (LQR). Our motivation for solving that problem using iterative gradient based techniques is that our basic parallel approach can be applied to more complex control problems in which the system dynamics can be linearized and the cost approximated quadratically. Furthermore, efficient parallel implementations of gradient methods such as best-step steepest descent and conjugate gradient suggests that similar parallel implementations of penalty function and gradient projection methods may be used to solve constrained control problems.

Our approach to the parallel evaluation of the step and gradient reduces the total number of operations required by sharing common terms when possible and then introduces parallelism. The degree of parallelism exhibited by the step and gradient computation techniques presented in this paper varies as a function of the number of processors to be used. We constrain the number of available processors, p, to lie in the range $1 \leq p \leq nN^{1/2}$, where n is the size of the system state vector, N is the number of stages in the control process and we assume $n \geq m$, where m is the size of the control. One of the features of the proposed parallel iterative algorithms is that their structure is completely specified by the number of processors whenever the number of stages $N \geq (p/n)^2$.

An efficient technique for gradient evaluation using a single processor has been discussed by Polak [14, pp.66-69]. A direct implementation of this technique on p processors achieves linear speedup for p up to n; however, for $p > n$, the speedup is significantly reduced. In this paper, we present an approach to gradient computation which reduces to a direct parallel implementation of the technique given in [8] when $1 \leq p \leq n$ and achieves speedup greater than $1/2(p+n)$ when $n < p \leq nN^{1/2}$.

A critical step in our approach involves the parallel computation of the state and costate vectors. When $n = 1$, the computations reduce to solving forward and reverse linear recurrence systems, both of size N. The parallel evaluation of m-th order linear recurrence systems has been extensively studied [1]-[3], [7]. To solve first-order linear block recurrence systems in parallel, we use a blocked formulation of the approach presented in [7].

The organization of this paper is as follows : in Section II, we state the unconstrained discrete linear quadratic optimal control problem, examine the gradient of the cost function and give the steepest descent algorithm we shall consider. Section III presents the step length and gradient computations required at each iteration. In Section IV we give a parallel procedure to solve the linear recurrence systems required by Section III. Based upon the results of Sections III and IV, Section V presents parallel implementations of the best-step steepest descent method to solve the LQR problem and the corresponding performance analysis. Finally, in Section VI conclusions are presented.

II. PRELIMINARIES

We consider the LQR discrete optimal control problem:

Problem 1: Given an m-input, discrete, time-varying linear system in which we are given the initial state, z_0, and

$$z_i = A_i z_{i-1} + B_i u_i, \quad i = 1,2,\ldots N, \tag{1}$$

where for $i = 0,1,\ldots,N$, z_i in E^n is the state of the system at time i and for $i = 1,2,\ldots,N$, u_i in E^m is the control at time i, find the mN control vector $u = (u_1^t, u_2^t, \ldots, u_N^t)^t$ that minimizes the performance index

$$J(u) = \frac{1}{2}\left(z^t Q z + u^t R u\right),$$

where z is the nN vector $z = (z_1^t, z_2^t, \ldots, z_N^t)^t$, $Q = Diag(Q_1, Q_2, \ldots, Q_N)$ is the $nN \times nN$ block matrix that consists of N $n \times n$ symmetric positive semi-definite blocks Q_i and $R = Diag(R_1, R_2, \ldots, R_N)$ is the $mN \times mN$ block matrix that consists of N $m \times m$ symmetric positive definite blocks R_i.

The hypotheses on the matrices Q and R insure that Problem 1 possesses a unique solution u^* and that $(dJ/du)_{u=u^*} = 0$.

We now introduce a formulation for dJ/du that is used in our parallel implementation of gradient algorithms.

A direct application of the chain rule for differentiation yields

$$\frac{dJ}{du} = \frac{\partial J}{\partial u} + \frac{\partial J}{\partial z}\frac{dz}{du},$$

where the z_i's are determined in accordance with Eq. (1), $\partial J/\partial u$ and $\partial J/\partial z$ are the $1 \times mN$ and $1 \times nN$ Jacobian matrices $u^t R$ and $z^t Q$ respectively, and dz/du is the $nN \times mN$ block lower triangular Jacobian matrix that consists of N^2 $n \times m$ blocks $(dz/du)_{ij} = dz_i/du_j$ obtained by the chain rule for all i and j in $[1,2,\ldots,N]$ by

$$\frac{dz_i}{du_j} = \begin{cases} 0 & \text{if } i < j \\ B_j & \text{if } i = j \\ A_i A_{i-1} \cdots A_{j+1} B_j & \text{if } i > j. \end{cases} \tag{2}$$

Eq. (2) shows that the influence matrix dz/du satisfies $F_z(dz/du) = F_u$, where F_z is the $nN \times nN$ block lower bidiagonal

This work was supported by the Air Force Office of Scientific Research under Contract AFOSR-85-0097.

matrix that consists of N^2 $n \times n$ blocks $\left(F_z\right)_{ij}$ and F_u is the $nN \times mN$ block diagonal matrix that consists of N^2 $n \times m$ blocks $\left(F_u\right)_{ij}$ defined for all i and j in $[1,2,\ldots,N]$ by

$$\left(F_z\right)_{ij} = \begin{cases} I & \text{if } i = j \\ -A_i & \text{if } i = j+1, \\ 0 & \textit{otherwise} \end{cases} \qquad \left(F_u\right)_{ij} = \begin{cases} B_i & \text{if } i = j \\ 0 & \textit{otherwise.} \end{cases}$$

Let F_0 be the $nN \times n$ block matrix that consists of N $n \times n$ blocks $\left(F_0\right)_i$ defined for all i in $[1,2,\ldots,N]$ by

$$\left(F_0\right)_i = \begin{cases} A_1 & \text{if } i = 1 \\ 0 & \textit{otherwise,} \end{cases}$$

let $g = dJ/du^t$ be the the $mN \times 1$ gradient of $J(u)$ with respect to u and let λ be the nN costate vector $\lambda = (\lambda_1^t, \lambda_2^t, \ldots, \lambda_N^t)^t$, λ_i in E^n, defined by $\lambda = F_z^{-t}Qz$. Then, given u and z_0, the gradient g may be obtained by using the three equations

$$F_z z = F_u u + F_0 z_0, \tag{3}$$

$$F_z^t \lambda = Qz, \tag{4}$$

$$g = Ru + F_u^t \lambda \tag{5}$$

With the notation $g^k = (dJ/du)^t_{u=u^k}$, the version of the best-step steepest descent method that we use is the following:

Algorithm 1: Let u^1 be given.

Step 0: Set $k = 1$ and compute g^1.

Step 1: If $\|g^k\|^2 \le \epsilon$ stop; else go to Step 2.

Step 2: Compute α^k to minimize $J(u^k - \alpha^k g^k)$.

Step 3: Compute g^{k+1}.

Step 4: Let $u^{k+1} = u^k - \alpha^k g^k$.

Step 5: Set $k = k + 1$ and go to Step 1.

In Section III we present our approach to the computation of g^1, α^k, and g^{k+1} and we show that the computation of α^k and g^{k+1} shares common terms.

III. GRADIENT AND BEST STEP COMPUTATION

We first consider the computation of g^1, which is performed only once in Step 0 of Algorithm 1. As a consequence of Eqs. (3), (4) and (5) we obtain the gradient evaluation technique proposed by Polak [8]:

Algorithm 2: Given u^1 and z_0.

Step 1: Compute z^1 such that $F_z z^1 = F_u u^1 + F_0 z_0$.

Step 2: Compute λ^1 such that $F_z^t \lambda^1 = Qz^1$.

Step 3: Compute $g^1 = Ru^1 + F_u^t \lambda^1$.

Due to the lower $n \times n$ block bidiagonal structure of F_z, Steps 1 and 2 of Algorithm 2 require the solution of N stage forward and reverse $n \times n$ block first-order linear recurrences, respectively. Parallel procedures for linear recurrences are presented in next section. Given λ^1, Step 3 computes g^1 by computing each of the N uncoupled components $g_i^1 = (dJ/du_i)^t_{u=u^1}$; thus, Step 3 exhibits linear speedup when executed in parallel.

We now consider the computation of the optimal step length α^k. The cost function is quadratic and therefore a closed form solution for α^k exists. It is clear that

$$J(u) = \frac{1}{2}u^t(R + F_u^t F_z^{-t} Q F_z^{-1} F_u)u$$

$$+ z_0^t F_0^t F_z^{-t} Q F_z^{-1} F_u u + \frac{1}{2} z_0^t F_0^t F_z^{-t} Q F_z^{-1} F_0 z_0$$

therefore

$$J(u^k - \alpha g^k) = a\alpha^2 + b\alpha + c,$$

where

$$a = \frac{1}{2}\langle g^k, d^k \rangle,\ b = -\langle g^k g^k \rangle,\ c = J(u^k),$$

$$d^k = Rg^k + F_u^t F_z^{-t} Q F_z^{-1} F_u g^k,$$

and it follows that the optimal step length α^k is

$$\alpha^k = -\frac{b}{2a} = \frac{\langle g^k g^k \rangle}{\langle g^k, d^k \rangle}.$$

Once d^k is known, the gradient g^{k+1} is easy to evaluate using

$$g^{k+1} = g^k - \alpha^k d^k.$$

Our approach to computing the quantity d^k consists of using Algorithm 3 below, where we note that instead of requiring F_z^{-1} and F_z^{-t}, we solve linear systems corresponding to F_z and F_z^t.

Algorithm 3: Given g^k.

Step 1: Compute $\mu^k = Rg^k$.

Step 2: Compute $\delta^k = F_u g^k$.

Step 3: Compute w^k such that $F_z w^k = \delta^k$.

Step 4: Compute $\eta^k = Qw^k$.

Step 5: Compute v^k such that $F_z^t v^k = \eta^k$.

Step 6: Compute $\chi^k = F_u^t v^k$.

Step 7: Compute $d^k = \mu^k + \chi^k$.

With the exception of Steps 3 and 5, Algorithm 3 computes d^k by executing a series of matrix-vector products followed by a vector sum. Each of the matrix-vector products consists of N uncoupled block matrix-vector products, thus exhibiting linear speedup when implemented in parallel. However, due to the structure of F_z, Steps 3 and 5 require the solution of N stage forward and reverse $n \times n$ block first-order linear recurrences, respectively. As in the case of Algorithm 2, this again suggests the need for parallel procedures to solve linear recurrences. Note that the state z^{k+1} and costate λ^{k+1} corresponding to u^{k+1} can be obtained easily from the quantities w^k and v^k computed in Steps 3 and 5 of Algorithm 3, that is,

$$z^{k+1} = z^k - \alpha^k w^k \quad \text{and} \quad \lambda^{k+1} = \lambda^k - \alpha^k v^k.$$

IV. PARALLEL PROCEDURES FOR LINEAR RECURRENCES

The model of SIMD parallel computation that we use consists of a global parallel memory, p parallel processors, and a control unit, where all processors perform the same operation at each time step. We further simplify the model by making the following assumptions: (i) each computational operation takes the same amount of time, referred to as a step, (ii) there are no accessing conflicts in global memory, (iii) all initial data resides in global memory, and (iv) there is no time required to access global memory.

We now present a blocked version of the parallel scalar approach given in [7] to solve forward N stage $n \times n$ block first-order linear recurrence systems that we use to implement Algorithms 2 and 3.

The forward recurrence problem is: given $n \times n$ matrices A_i, $i = 2, 3, \ldots, N$ and given vectors $\gamma_i \in E^n$, $i = 1,2,\ldots,N$, find the n vectors z_i such that $z_1 = \gamma_1$ and $z_i = A_i z_{i-1} + \gamma_i$, $i = 2, 3, \ldots, N$. Let

$$\Omega = \begin{cases} \dfrac{N-1}{(p/n)^2-1)} & \text{if } p > n \\ N-1 & \textit{otherwise,} \end{cases} \qquad \kappa = \begin{cases} p/n & \text{if } p > n \\ 1 & \textit{otherwise.} \end{cases}$$

For ω in $[0,1,\ldots,\Omega-1]$, define the index sets

$$f_0(\omega) = \{\kappa i + \omega(\kappa^2-1) : i = \kappa-1, \kappa-2, \ldots, 1\}$$

and

$$f_1(\omega) = \{\kappa i + Max(\omega+1, \omega(\kappa^2-1)) : i = \kappa, \kappa-1, \ldots, 1\}.$$

Thus, given $\gamma = (\gamma_1^t, \gamma_2^t, \ldots, \gamma_N^t)^t$, $\gamma_i \in E^n$ and precomputed $A[i+j, j] = A_{i+j} A_{i+j-1} \cdots A_j$, $j \in f_0(\omega)$, i in $[0,1,\ldots,\kappa-1]$, the following procedure solves the forward block recurrence system, where for presentation simplicity, we assume that Ω and κ are integers.

```
1. PROCEDURE FORWARD($N,n,p,\gamma$)
2.  $z_1 := \gamma_1$
3.  FOR $\omega := 0$ TO $\Omega - 1$ DO
4.    FORALL $j \in f_0(\omega)$ DO IN PARALLEL $z_j := \gamma_j$;
5.    FOR $i := 1$ TO $Max(1,\kappa-1)$ DO
6.      FORALL $j \in f_1(\omega)$ DO IN PARALLEL
7.        $z_{i+j} := A_{i+j} z_{i+j-1} + \gamma_{i+j}$;
8.      END FORALL
9.    END FOR
10.   FORALL $j \in f_0(\omega)$ DO
11.     FOR $i := 0$ TO $\kappa-1$ DO IN PARALLEL
12.       $z_{i+j} := A[i+j,j]z_{j-1} + z_{i+j}$;
13.     END FOR
14.   END FORALL
15. END FOR
16. END PROCEDURE
```

When $1 \le p \le n$, the index set $f_0(\omega)$ is empty and procedure FORWARD reduces to sequentially executing step 7 $N-1$ times, each execution using p processors. When $n < p \le nN^{1/2}$, procedure FORWARD sequentially solves Ω reduced block recurrence systems in z_i of size $(p/n)^2$, each in parallel. Each reduced system is solved in two phases: the first phase consists of the execution of steps 4 and 5 and computes $(p/n)^2$ partial solutions, in which the first p/n are the actual solutions and the second phase consists of the execution of loop 10 in which the precomputed $n \times n$ block matrices $A[i+j,j]$ are used to update the next p/n partial solutions at each iteration. We assign n processors to perform each of the p/n concurrent executions of steps 7 and 12. The complete solution to the block recurrence system is obtained after executing Ω iterations of loop 3. If Ω is not an integer, then we replace Ω by $\lceil \Omega \rceil$ and simply terminate the computation when z_N is computed and if κ is not an integer, we replace κ by $\lfloor p/n \rfloor$.

A similar procedure REVERSE used to solve reverse recurrences in parallel may be obtained by a straightforward modification of procedure FORWARD and hence will not be given.

We now give the number of steps required to solve either an N stage forward or reverse first-order $n \times n$ block linear recurrence system.

Theorem 1: Given N, n and p such that $p = 1$ or p/n is an integer, the number of parallel steps required to solve a block $n \times n$ first-order linear recurrence system of length N using p processors is

$$T_p = \begin{cases} (N-1)\dfrac{2n^2}{p} & \text{if } 1 \le p \le n \\ \lceil \Omega \rceil 4(p-n) & \text{if } n < p \le nN^{1/2}. \end{cases}$$

It is clear from Theorem 1 that the speedup $S_p = T_1/T_p$ exhibited by the procedures FORWARD and REVERSE is p when $1 \le p \le n$ and $1/2(p+n)$ when $n < p \le nN^{1/2}$ and p/n and Ω are integers. The corresponding efficiency $E_p = S_p/p$ is therefore 1 when $1 \le p \le n$ and $1/2 + n/2p$ when $n < p \le nN^{1/2}$ and p/n and Ω are integers. In Figs. 1, 2, 3 and 4 we plot E_p for the values of $n = 8$, 16, 32 and 64 respectively, where the efficiency corresponding to the procedures FORWARD and REVERSE is denoted by the solid line in each plot. Thus, we see that the efficiency increases with increasing values of n and p and is independent of N.

V. PARALLEL BEST STEP STEEPEST DESCENT

We now use the parallel procedures for the solution of linear recurrences discussed in the previous section to obtain parallel implementations of Algorithms 2 and 3 give the corresponding number of steps required for their execution when p processors are used.

We first give the parallel implementation of Algorithm 2.

```
1. PROCEDURE GRADIENT($z_0,u^1$)
2.  FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $\gamma_i^1 := B_i u_i^1$;
3.  $\gamma_1^1 := \gamma_1^1 + A_1 z_0$;
4.  $z^1$ = FORWARD($N,n,p,\gamma^1$);
5.  FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $\zeta_i^1 := Q_i z_i^1$;
6.  $\lambda^1$ = REVERSE($N,n,p,\zeta^1$);
7.  FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $g_i^1 := R_i u_i^1 + B_i^t \lambda_i^1$;
8.  RETURN $g^1$;
9. END PROCEDURE
```

Lemma 1: Given z_0, u^1, N, n, m and p such that $p = 1$ or p/n is an integer, the number of steps required by the procedure GRADIENT to compute g^1 using p processors, $1 \le p \le nN^{1/2}$, is

$$T_p = \begin{cases} \dfrac{Nn}{p}(6n+2m-2) + \left\lceil \dfrac{Nm}{p} \right\rceil (2n+2m-1) - \dfrac{2n^2}{p} & \text{if } 1 \le p \le n \\ \left\lceil \dfrac{Nn}{p} \right\rceil (2n+2m-2) + \left\lceil \dfrac{Nm}{p} \right\rceil (2n+2m-1) + \lceil \Omega \rceil 8(p-n) + 2n & \text{if } n < p \le nN^{1/2} \end{cases}$$

We next give the parallel implementation of Algorithm 3.

```
1. PROCEDURE DIRECTION($g^k$)
2.  FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $\mu_i^k := R_i g_i^k$;
3.  FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $\delta_i^k := B_i g_i^k$;
4.  $w^k$ = FORWARD($N,n,p,\delta^k$);
5.  FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $\eta_i^k := Q_i w_i^k$;
6.  $v^k$ = REVERSE($N,n,p,\eta^k$);
7.  FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $\chi_i^k := B_i^t v_i^k$;
8.  FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $d_i^k := \mu_i^k + \chi_i^k$;
9.  RETURN $d^k$;
10. END PROCEDURE
```

Lemma 2: Given g^k, N, n, m and p such that $p = 1$ or p/n is an integer, the number of steps required by procedure DIRECTION to compute d^k using p processors, $1 \le p \le nN^{1/2}$, is

$$T_p = \begin{cases} \dfrac{Nn}{p}(6n+2m-2) + \left\lceil \dfrac{Nm}{p} \right\rceil (2n+2m-1) - \dfrac{4n^2}{p} & \text{if } 1 \le p \le n \\ \left\lceil \dfrac{Nn}{p} \right\rceil (2n+2m-2) + \left\lceil \dfrac{Nm}{p} \right\rceil (2n+2m-1) + \lceil \Omega \rceil 8(p-n) & \text{if } n < p \le nN^{1/2} \end{cases}$$

We now embed the parallel procedures *GRADIENT* and DIRECTION to obtain a parallel implementation of Algorithm 1 and we then give the corresponding number of steps required for one iteration.

```
1. PROCEDURE PSDM($z_0,u^1$)
2.  $k = 1$;
3.  $g^1$ = GRADIENT($z_0,u^1$);
4.  WHILE $\|g^k\|^2 > \epsilon$ DO
5.    $d^k$ = DIRECTION($g^k$);
6.    $\alpha^k = \langle g^k,g^k \rangle / \langle g^k,d^k \rangle$;
7.    FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $g_i^{k+1} := g_i^k - \alpha^k d_i^k$;
8.    FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $u_i^{k+1} := u_i^k - \alpha^k g_i^k$;
9.    $k = k + 1$;
10.  END WHILE
11. END PROCEDURE
```

Theorem 2: Given z_0, u^1, N, n, m and p such that $p = 1$ or p/n is an integer, the number of steps required by one iteration of procedure PSDM using p processors, $1 \le p \le nN^{1/2}$, is

$$T_p = \begin{cases} \left\lceil \dfrac{Nn}{p} \right\rceil (6n+2m-2) + \left\lceil \dfrac{Nm}{p} \right\rceil (2n+2m+7) - \dfrac{4n^2}{p} + 2\log_2 p & \text{if } 1 \le p \le n \\ \left\lceil \dfrac{Nn}{p} \right\rceil (2n+2m-2) + \left\lceil \dfrac{Nm}{p} \right\rceil (2n+2m+7) + \lceil \Omega \rceil 8(p-n) + 2\log_2 p & \text{if } n < p \le nN^{1/2} \end{cases}$$

As a consequence of Theorem 2, it may be shown that the speedup S_p and efficiency E_p for procedure PSDM are bounded from below by

$$S_p = \frac{T_1}{T_p} \ge 0.6p, \quad E_p = \frac{T_1}{pT_p} \ge 0.6$$

In Figs. 1, 2, 3 and 4, we plot E_p for the procedure PSDM for the values $n = 8$, 16, 32 and 64 respectively, and in each case use the values $m = 1$, 4 and 8. It is then easy to see that E_p increases with m, and that the efficiency of the procedure PSDM is bounded from below by the efficiency of the procedures FORWARD and REVERSE.

VI. CONCLUSIONS

In this paper a parallel implementation of the best-step steepest descent method has been presented to solve the LQR optimal control problem. The procedure exhibits the desirable property that its structure, and hence parallelism, is determined by the number of available processors. Thus, unlike approaches in which the structure of the procedure changes with problem size, the procedure presented in this paper maintains the same computational and interprocessor communication requirements, independently of the number of stages in the control problem. Furthermore, the procedure has been shown to exhibit an efficiency E_p always greater than 0.6.

The paper's basic approach can be used to produce parallel implementations of more complex gradient based methods. For example, the procedure PSDM may be easily modified to produce the following parallel version of the Fletcher-Reeves conjugate gradient method.

```
1. PROCEDURE PCGM($z_0,u^1$)
2.   $k = 1$;
3.   $\pi^1 = g^1 =$ GRADIENT($z_0,u^1$);
4.   WHILE $\|g^k\|^2 > \epsilon$ DO
5.     $d^k =$ DIRECTION($\pi^k$);
6.     $\alpha^k = <g^k,\pi^k>/<d^k,\pi^k>$;
7.     FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $g_i^{k+1} := g_i^k - \alpha^k d_i^k$;
8.     FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL $u_i^{k+1} := u_i^k - \alpha^k \pi_i^k$;
9.     $\beta^k = <g^{k+1},g^{k+1}>/<g^k,g^k>$;
10.    FORALL $i \in \{1,2,...,N\}$ DO IN PARALLEL
                                      $\pi_i^{k+1} := g_i^{k+1} - \beta^k \pi_i^k$;
11.    $k = k + 1$;
12.  END-WHILE
13. END PROCEDURE
```

Finally, we note that the procedures presented in this paper may be used to solve discrete optimal control problems which involve nonquadratic cost, nonlinear dynamics and constraints on the states and/or controls. In that case, one needs to use penalty function and gradient projection methods and suitable approximations to cost function and system dynamics.

REFERENCES

[1] Chen, S.C. and Kuck, D.J., "Time and Parallel Processor Bounds for Linear Recurrence Systems", *IEEE Trans. Computers*, Vol. C-24, No.7, July 1975, pp. 701-717.

[2] Chen, S.C., Kuck, D.J. and Sameh, A.H., "Practical Parallel Band Triangular System Solvers", *ACM Trans. on Mathematical Software*, Vol. 4, No. 3, September 1978, pp. 270-277.

[3] Gajski, D. J., "An Algorithm for Solving Linear Recurrence Systems on Parallel and Pipelined Machines", *IEEE Trans. Computers*, Vol. C-30, No. 3, March 1981, pp. 190-206.

[4] Larson, R.E. and Edison, T., Parallel Processing Algorithms for the Optimal Control of Nonlinear Dynamic Systems", *IEEE Trans. Computers*, Vol. C-22, No. 8, August 1973, pp. 777-786.

[5] Lewis, F. L., *Optimal Control*, John Wiley and Sons, New York, N.Y., 1986.

[6] Meyer, G.G.L., "A Segmented Algorithm for Solving a Class of State Constrained Discrete Optimal Control Problems", *Decision and Control Theory Conference*, San Diego, California, 1973, pp. 73-79.

[7] Meyer, G.G.L., and Podrazik, L.J., "A Parallel First-Order Linear Recurrence Solver", *Proceedings of the 20-th Annual Conference Information Sciences & Systems*, Princeton, New Jersey, March 19-21, 1986, also to appear in the *Journal of Parallel and Distributed Computing*.

[8] Polak, E., *Computational Methods in Optimization, A Unified Approach*, Academic Press, New York and London, 1971.

[9] Scheel, C. and McInnis, B., "Parallel Processing of Optimal Control Problems by Dynamic Programming", *Information Sciences*, Vol. 25, No. 2, November 1981, pp. 85-114.

Figure 1 - Efficiency of Procedures FORWARD, REVERSE and PSDM

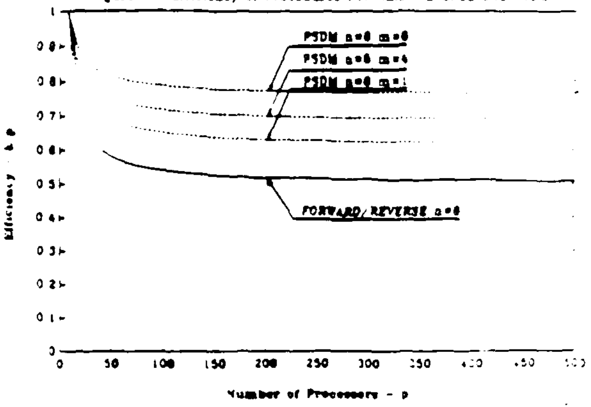


Figure 2 - Efficiency of Procedures FORWARD, REVERSE and PSDM

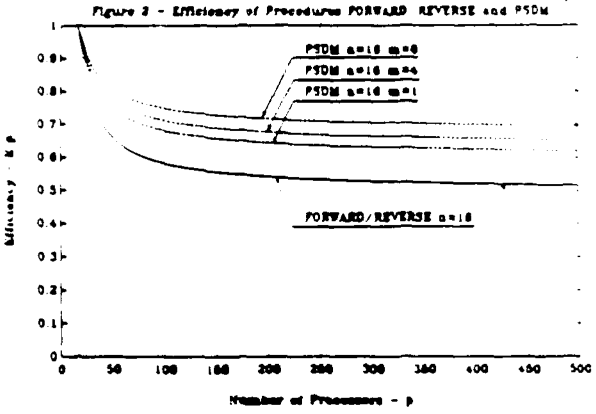


Figure 3 - Efficiency of Procedures FORWARD, REVERSE and PSDM

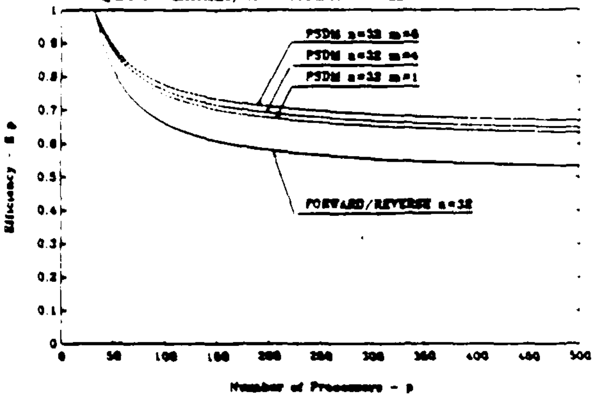
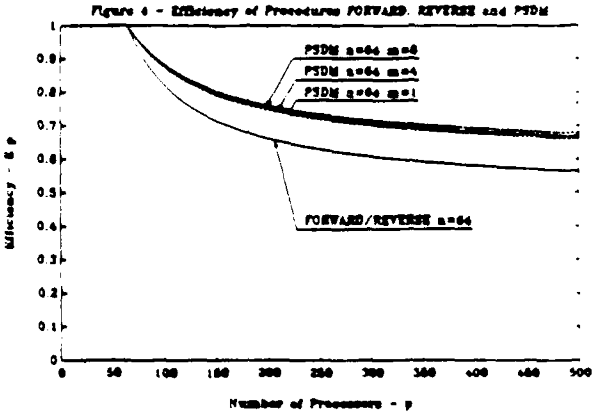


Figure 4 - Efficiency of Procedures FORWARD, REVERSE and PSDM



8. APPENDIX III

PARALLEL GRADIENT PROJECTION ALGORITHMS TO SOLVE THE DISCRETE LQR OPTIMAL CONTROL PROBLEM WITH HARD CONTROL BOUNDS

GERARD G. L. MEYER
Electrical and Computer Engineering Department
The Johns Hopkins University
Baltimore, Maryland 21218

LOUIS J. PODRAZIK
Bendix Environmental Systems Division
Allied-Signal Inc.
Baltimore, Maryland 21284-9840

ABSTRACT

In this paper we present two parallel gradient based iterative algorithms to solve the linear quadratic regulator (LQR) optimal control problem with hard control bounds. In the first part of the paper, we introduce the algorithms in the context of the general class of problems to which they are applicable. The first algorithm is a parametrized gradient projection method and can be used to solve any convex programming problem. The second algorithm is a combination of the first algorithm with a constrained version of the Fletcher-Reeves conjugate gradient method and can be used to solve linear inequality constrained problems. We then use the two algorithms to solve the LQR optimal control problem with hard control bounds. In the second part of the paper, we show that at each iteration parallel evaluation of the step length and projected gradient of the quadratic cost function can be efficiently performed as a function of the number of processors. We then embed our parallel step length and gradient projection procedures to produce two parallel algorithms which are suitable for real-time online implementation on a SIMD machine.

I. INTRODUCTION

Practical iterative methods to solve optimal control problems must exhibit fast convergence coupled with low computational overhead per iteration so that they may be implemented in a real-time online environment. Furthermore, they must be interior descent methods since infeasible approximations to the solution are unacceptable for most applications. In this paper we present two gradient based iterative interior methods for the parallel solution of the discrete LQR optimal control problem with hard control bounds. The algorithms are synthesized so that each iteration may be efficiently executed on a parallel computer. Unlike previous parallel approaches to the solution of optimal control problems which simply fold the computations to fit the number of processors [MEY73], [SCH81], [TRA80], our approach has been devised by explicitly considering the given computational environment. Consequently, one of the features of the parallel algorithms presented in this paper is that their structure, and hence their parallelism, is determined by the number of available processors, resulting in algorithms which are matched to the given parallel environment.

We introduce the algorithms in the context of the general class of problems to which they are applicable. The first algorithm is a parametrized gradient projection method and can be used to solve any convex programming problem. We parametrize the algorithm for two purposes: first, the algorithm contains a so called ϵ-procedure for determining the active constraints in order to prevent the possibility of jamming and to ensure convergence; unlike approaches which generate a sequence of ϵ's, our approach uses a constant ϵ for all iterations. Secondly, we investigate the performance of the algorithm for various parameter choices with the goal of obtaining improved convergence. The second algorithm is a combination of the first algorithm with a constrained version of the Fletcher-Reeves conjugate gradient method and can be used to solve any problem with linear inequality constraints. Furthermore, we show that a slight modification of our second algorithm results in the algorithm exhibiting finite convergence on problems

Supported by the Air Force Office of Scientific Research under Contract AFOSR-85-0097.

with quadratic cost.

We then use c algorithms to solve the LQR optimal cont. problem with hard control bounds which we rewrite as a bounded variable quadratic programming problem with special structure. There exist many interior methods applicable to this class of problems, including feasible directions, gradient projection, reduced gradient, variable metric and numerous quadratic programming methods. Our motivation for devising a gradient projection method is due to the simple projection computation that occurs with respect to the control problem of interest as well as avoiding large matrix computations involving the exact or approximated Hessian.

We give two parallel implementations of our algorithms to solve the LQR optimal control problem with hard control bounds. In both implementations the step length and gradient is obtained by using the parallel procedures presented in [MEY87b]. The procedures share common terms and exhibit varying degrees of parallelism as a function of the number of processors. We constrain the number of available processors, p, to lie in the range $1 \leq p \leq nN^{1/2}$, where n is the size of the system state vector, N is the number of stages in the control process and we assume $n \geq m$, where m is the size of the control.

By employing a blocked formulation of the approach for first-order linear recurrences given in [MEY87a], we embed in our algorithms the approach for gradient computation given in [MEY87b] which reduces to a direct parallel implementation of the technique given in [POL71, pp.66-69] when $1 \leq p \leq n$ and achieves speedup greater than $1/2(p + n)$ when $n < p \leq nN^{1/2}$. In addition, one of the features of our proposed parallel iterative algorithms is that their structure is completely specified by the number of processors whenever the number of stages $N \geq (p/n)^2$.

The organization of this paper is as follows : Section II introduces the convex problem and associated optimality conditions. Section III presents the parametrized gradient projection algorithm and corresponding implementation for the bounded variable quadratic programming problem. In Section IV we combine the algorithm given in Section III with a linearly constrained version of the Fletcher-Reeves conjugate gradient method. We then rewrite the LQR problem with hard control bounds as a bounded variable quadratic programming problem in Section V. Section VI presents the parallel algorithms to solve the LQR problem with hard control bounds and the corresponding performance analysis. Finally, in Section VII conclusions are presented.

II. THE CONVEX PROBLEM AND PRELIMINARIES

We consider the following problem:

Problem 1: Given $m+1$ maps $f^0(.), f^1(.), ..., f^m(.) : E^n \rightarrow E^1$ and a subset Ω of E^n defined by

$$\Omega = \{x \mid f^i(x) \leq 0 \text{ for } i = 1, 2, ..., m\},$$

find a point x^* in Ω such that for every x in Ω

$$f^0(x^*) \leq f^0(x).$$

Definition 1: Let C be the class of all problems of the form of Problem 1 in which the maps $f^0(.), f^1(.), ..., f^m(.)$ are such that

(i) $f^0(.)$ is continuously differentiable, convex and radially unbounded; that is, $f^0(.)$ is such that given any $x \in \Omega$, to every scalar α corresponds a scalar $\rho > 0$ such that $f^0(x) > \alpha$ whenever $||x|| > \rho$,

(ii) $f^i(.)$, $i = 1, 2, ..., m$, are continuously differentiable, convex and define a nonempty constraint set Ω and

(iii) the set Ω satisfies the Kuhn-Tucker constraint qualification at every solution x^*.

It is known that the following necessary conditions of optimality are also sufficient for any problem in C.

Optimality Conditions: A point $x^* \in \Omega$ is an optimal solution to a problem in C if and only if there exists a vector $\mu \in E^m$ with components $\mu_i \geq 0$, $i = 1, 2, ..., m$, such that

$$\nabla f^0(x^*)^t + \sum_{i=1}^{m} \mu_i \nabla f^i(x^*)^t = 0$$

$$\mu_i f^i(x^*) = 0, \text{ for } i = 1, 2, \ldots, m.$$

III. THE PARAMETRIZED GRADIENT PROJECTION ALGORITHM

Given a point $x \in \Omega$ and the parameter $\epsilon \geq 0$, define the ϵ-active constraints index set $I(x,\epsilon)$ as

$$I(x,\epsilon) = \{i \mid f^i(x) \geq -\epsilon\},$$

given a subset I of the set $\{1, 2, \ldots, m\}$, define the subset $\Omega(I)$ of E^n as

$$\Omega(I) = \begin{cases} \{y \mid f^i(y) \leq 0 \text{ for all } i \in I\} & \text{if } I \neq \phi \\ E^n & \text{if } I = \phi, \end{cases}$$

and given a point $x \in \Omega$ and an index set I, let $w(x,I)$ be the projection of $x - \nabla f^0(x)^t$ onto the set $\Omega(I)$, that is, $w(x,I)$ satisfies

$$\|(x - \nabla f^0(x)^t) - w(x,I)\| = \min_y\{\|(x - \nabla f^0(x)^t) - y\| \mid y \in \Omega(I)\}.$$

We only consider problems in C, and thus $\Omega(I)$ is non-empty and convex, and the quantity $w(x,I)$ is well defined. Using the notation just defined, we may rewrite the optimality conditions as:

Lemma 1: If a point $x^* \in \Omega$ is optimal for a problem in C then $w(x^*, I(x^*,\epsilon)) = x^*$ for all $\epsilon \geq 0$. Conversely, if x^* and ϵ satisfy $w(x^*, I(x^*,\epsilon)) = x^*$, then x^* is optimal.

We now give the parametrized gradient projection algorithm to solve a Problem in the class C.

Algorithm 2: Let $x^1 \in \Omega$ and $\epsilon \geq 0$ be given.

```
Step 0: Set $k = 1$.
        Compute $\nabla f^0(x^1)$, $I(x^1,\epsilon) = \{i \mid f^i(x^1) \geq -\epsilon\}$ and $p^1 = w(x^1, I(x^1,\epsilon)) - x^1$.
Step 1: If $p^k = 0$ stop; else go to Step 2.
Step 2: Set $d^k = p^k$.
Step 3: Compute $\alpha^k > 0$ to minimize $f^0(x^k + \alpha^k d^k)$ subject to $(x^k + \alpha^k d^k) \in \Omega$.
Step 4: Let $x^{k+1} = x^k + \alpha^k d^k$.
Step 5: Compute $\nabla f^0(x^{k+1})$.
Step 6: Compute $I(x^{k+1},\epsilon) = \{i \mid f^i(x^{k+1}) \geq -\epsilon\}$.
Step 7: Compute $p^{k+1} = w(x^{k+1}, I(x^{k+1},\epsilon)) - x^{k+1}$.
Step 8: Set $k = k + 1$ and go to Step 1.
```

Remark 1: Although not explicitly required in the presentation of Algorithm 2, we introduce the quantity p^k in order to facilitate a later modification which results in our second algorithm.

At each iteration, Algorithm 2 generates a search direction d^k which is computed by projecting the negative gradient onto the set $\Omega(I(x^k,\epsilon))$. The parameter ϵ defines the "sufficiently-active" constraint region. If a point x^k is in that region, then $I(x^k,\epsilon) \neq \phi$ and the corresponding search direction d^k is obtained by projecting $x^k - \nabla f^0(x^k)^t$ onto the set $\{y \mid f^i(y) \leq 0 \text{ for all } i \in I(x^k,\epsilon)\}$; otherwise, $d^k = -\nabla f^0(x^k)^t$. For ϵ large enough, $\Omega(I(x^k,\epsilon)) = \Omega$ for all k. Thus, by choosing ϵ large, Algorithm 2 reduces to projecting $x^k - \nabla f^0(x^k)^t$ onto the entire set Ω at each iteration. For $\epsilon = 0$, $\Omega(I(x^k,\epsilon)) = E^n$ whenever $x^k \in \text{interior}(\Omega)$. Thus, when $\epsilon = 0$, Algorithm 2 reduces to a projected version of best-step steepest descent in which the direction chosen at an interior point is the negative gradient.

We now show that using any $\epsilon > 0$ in Algorithm 2 will always produce a solution to a problem in C, even though the direction finding map used to compute d^k is not closed.

Lemma 2: If $x \in \Omega$ is [illegible] a solution to a given problem in C and ϵ [illegible] then there exists an $\epsilon(x) > 0$ and $\delta(x) > 0$ such that

$$f^0(y + \alpha d) \le f^0(y) - \delta(x) \text{ for all } y \in B(x, \epsilon(x)) \cap \Omega .$$

Theorem 1: If $\{x^k\}$ is a sequence constructed by Algorithm 2 to solve a problem in C then either $\{x^k\}$ is finite and the last point is optimal or $\{x^k\}$ is infinite and contains cluster points, each of which is optimal. Furthermore, when a problem in C is such that $f^0(.)$ is strictly convex, then that problem has a unique optimal solution x^*, and in that case $\{x^k\}$ converges to x^*.

In order to present the parallel implementation of Algorithm 2 used to solve the LQR problem, we first use it to solve a particular case of Problem 1, specifically the bounded variable quadratic programming problem given below.

Problem 2: Find $x \in E^n$ with components x_i, $i = 1, 2, ..., n$, that minimizes the performance index $f^0(x) = 1/2\, x^t Hx + b^t x + c$ subject to $x \in \Omega$, where H is an $n \times n$ symmetric positive definite matrix, b is an $n \times 1$ vector and Ω is the unit hypercube defined as $\Omega = \{x \mid \ |x_i| \le 1, \text{ for } i = 1, 2, ..., n \}$.

Since H is positive definite, the cost function $f^0(.)$ is strictly convex. The nonempty constraint set Ω is the intersection of the $2n$ linear inequalities $f^i(x) = x_i - 1 \le 0$ and $f^{i+n}(x) = -x_i - 1 \le 0$ for $i = 1, 2, ..., n$. Therefore, Problem 2 lies in C and by Theorem 1, Algorithm 2 will generate a sequence which converges to the unique solution x^*.

We now give some properties of implementing Algorithm 2 to solve Problem 2.

Lemma 3: Given x^k and $\nabla f^0(x^k)$, let $p^k = w(x^k, I(x^k,\epsilon)) - x^k$ for the case of Problem 2. Then each component p_i^k of p^k can be computed as

$$p_i^k = \begin{cases} +1 - x_i^k & \text{if } \quad i \in I(x^k,\epsilon) \text{ and } x_i^k - \nabla f_i^0(x^k)^t \ge 1 \\ -1 - x_i^k & \text{if } i+n \in I(x^k,\epsilon) \text{ and } x_i^k - \nabla f_i^0(x^k)^t \le -1 \\ -\nabla f_i^0(x^k)^t & \text{otherwise,} \end{cases} \tag{3.1}$$

where we denote $\nabla f_i^0(x^k)$ to be the i-th component of $\nabla f^0(x^k)$.

Remark 2: If $\epsilon \ge 2$ then $I(x^k,\epsilon) = \{1, 2, ..., 2n\}$ for all k. Consequently, $w(x^k, I(x^k,\epsilon)) = SAT(x^k - \nabla f^0(x^k)^t)$ and Algorithm 2 reduces to projecting onto the entire set Ω at each iteration.

Lemma 4: Given x^k and $\nabla f^0(x^k)$, let α^k minimize $f^0(x^k + \alpha^k d^k)$ subject to $(x^k + \alpha^k d^k) \in \Omega$ for the case of Problem 2. Then α^k can be computed as $\alpha^k = \min\{\alpha_u, \alpha_c\}$, where

$$\alpha_u = -\langle d^k, \nabla f^0(x^k)^t \rangle \,/\, \langle d^k, Hd^k \rangle, \tag{3.2}$$

$$\alpha_c = \min_{\alpha} \{\alpha \mid \alpha = \frac{sgn\,(d_i^k) - x_i^k}{d_i^k} \text{ for all } i \text{ such that } d_i^k \ne 0\}. \tag{3.3}$$

Remark 3: Since the cost function is quadratic and we are using the update $x^{k+1} = x^k + \alpha^k d^k$, the quantity $\nabla f^0(x^{k+1})^t$ can be updated using $\nabla f^0(x^{k+1})^t = \nabla f^0(x^k)^t + \alpha^k Hd^k$. Consequently, we need only compute the gradient directly in Step 0 of Algorithm 2, and then use the gradient update at each subsequent iteration.

We now give the following implementation of Algorithm 2 to solve Problem 2.

Algorithm 3: Let x^1 and $\epsilon > 0$ be given.

```
Step 0: Set $k = 1$.
        Compute $\nabla f^0(x^1)^t = Hx^1 + b$, $I(x^1,\epsilon) = \{i \mid x_i^1 - 1 \ge -\epsilon\} \cup \{i+n \mid -x_i^1 - 1 \ge -\epsilon\}$ and $p^1$ using Eq. (3.1).
Step 1: If $\|p^k\|^2 \le \epsilon$, then stop; else go to Step 2.
```

Step 2: Set $d^k = p^k$.

Step 3: Compute $\alpha \quad \min\{\alpha_u, \alpha_c\}$, according to Eqs. (3.2) and (3.3).

Step 4: Let $x^{k+1} = x^k + \alpha^k d^k$.

Step 5: Let $\nabla f^0(x^{k+1})^t = \nabla f^0(x^k)^t + \alpha^k H d^k$.

Step 6: Compute $I(x^{k+1},\varepsilon) = \{i \mid x_i^{k+1} - 1 \geq -\varepsilon\} \cup \{i+n \mid -x_i^{k+1} - 1 \geq -\varepsilon\}$.

Step 7: Compute p^{k+1} using Eq. (3.1).

Step 8: Set $k = k + 1$ and go to Step 1.

In Section IV we modify Algorithm 3 to obtain improved convergence properties when a problem in C is linearly constrained.

IV. FLETCHER-REEVES MODIFICATION FOR LINEARLY CONSTRAINED PROBLEMS

We now consider the following linearly constrained class of problems.

Definition 2: Let L be the class of all problems of the form of Problem 1 in which the maps $f^0(.), f^1(.), \ldots, f^m(.)$ are such that

(i) $f^0(.)$ is continuously differentiable, convex and radially unbounded,

(ii) $f^i(x) = \langle g^i, x \rangle - h_i$, for $i = 1, 2, \ldots, m$ define a nonempty constraint set Ω and

(iii) for any $x \in \Omega$, g^i for $i \in I(x, 0)$ are linearly independent.

Given $x^1 \in \Omega$, let $I(x^k,0)$ be the index set defined by $I(x^k,0) = \{i \mid \langle g^i, x \rangle - h_i = 0\}$ and let G_k be the corresponding matrix whose rows are $(g^i)^t$ for all $i \in I(x^k,0)$. The projection matrix P_k associated with the point x^k is $P_k = I - G_k^t(G_kG_k^t)^{-1}G_k$. Let $M(I(x^k,0))$ be the linear manifold defined by

$$M(I(x^k,0)) = \begin{cases} \{x \mid \langle g^i, x \rangle - h_i = 0 \text{ for all } i \in I(x^k,0)\} & \text{if } I(x^k,0) \neq \phi \\ E^n & \text{if } I(x^k,0) = \varphi. \end{cases}$$

Then we may compute the projection of $\nabla f^0(x^k)$ onto the set $M(I(x^k,0))$ as $\hat{p}^k = -P_k \nabla f^0(x^k)^t$.

Our approach to solving a problem in L is to approximate $f^0(.)$ as a quadratic and generate a new direction d^{k+1} which is conjugate with respect to the previous direction d^k whenever possible. We compute the initial direction $d^1 = p^1$ and subsequent directions d^k using $d^k = p^k + \beta^k d^{k-1}$, where $\beta^k = \|p^k\|^2 / \|p^{k-1}\|^2$. However, rather than projecting $x^k - \nabla f^0(x^k)^t$ onto the active contstraint set $M(I(x^k,0))$, we project onto the approximation set $\Omega(I(x^k,\varepsilon))$, that is, we compute $p^k = w(x^k, I(x^k,\varepsilon)) - x^k$. Consequently, we may produce a d^{k+1} which is conjugate to d^k whenever $P_{k+1} = P_k$ and $p^{k+1} = \hat{p}^{k+1}$; otherwise we restart the algorithm with $d^{k+1} = p^{k+1}$. Furthermore, if $f^0(.)$ is not quadratic, the algorithm should be restarted at least every $n - |I(x^{k+1},0)|$ steps as a spacer step to ensure global convergence.

We now give the parametrized gradient projection algorithm with the Fletcher-Reeves modification to solve a problem in the class L.

Algorithm 4: Let x^1 and $\varepsilon > 0$ be given.

Step 0: Set $k = \hat{k} = 1$ and $d^0 = 0$.
Compute $\nabla f^0(x^1)$, $I(x^1,\varepsilon) = \{i \mid f^i(x^1) \geq -\varepsilon\}$, $I(x^1,0) = \{i \mid f^i(x^1) = 0\}$ and $p^1 = w(x^1, I(x^1,\varepsilon)) - x^1$.

Step 1: If $p^k = 0$ then stop; else go to Step 2.

Step 2: If $d^{k-1} = 0$ then let $d^k = p^k$ and go to Step 5; else go to Step 3.

Step 3: Let $\beta^k = \dfrac{\|p^k\|^2}{\|p^{k-1}\|^2}$.

Step 4: Let $d^k = p^k + \beta^k d^{k-1}$.

Step 5: Compute $\alpha^k > 0$ to minimize $f^0(x^k + \alpha^k d^k)$ subject to $(x^k + \alpha^k d^k) \in \Omega$.

Step 6: Let $x^{k+1} = x \quad \alpha^k d^k$.

Step 7: Compute $\nabla f^0(x^{k+1})$.

Step 8: Compute $I(x^{k+1},\epsilon) = \{i \mid f^i(x^{k+1}) \geq -\epsilon\}$.

Step 9: Compute $p^{k+1} = w(x^{k+1}, I(x^{k+1},\epsilon)) - x^{k+1}$.

Step 10: Compute $I(x^{k+1},0) = \{i \mid f^i(x^{k+1}) = 0\}$.

Step 11: Compute $\hat{p}^{k+1} = -P_{k+1}\nabla f^0(x^{k+1})^t$.

Step 12: If $I(x^{k+1},0) \neq I(x^k,0)$ or $p^{k+1} \neq \hat{p}^{k+1}$ or $\hat{k} > n - |I(x^{k+1},0)|$ then set $d^k = \hat{k} = 0$.

Step 13: Set $k = k + 1$, $\hat{k} = \hat{k} + 1$ and go to Step 1.

When $f^0(.)$ is a quadratic, Algorithm 4 generates a new direction d^{k+1} which is conjugate to d^k whenever the point x^{k+1} is in the same active constraint set $M(I(x^k,0)$ as x^k and $p^{k+1} = \hat{p}^{k+1}$. However, since p^{k+1} is obtained by projecting onto the set $\Omega(I(x^{k+1},\epsilon))$ rather than $M(I(x^k,0))$, Algorithm 4 is not a pure active set method. Thus, Algorithm 4 may change constraint sets before the minimizer on the current constraint set is found and consequently does not exhibit finite convergence in the case of quadratic cost.

Theorem 3: If $\{x^k\}$ is a sequence constructed by Algorithm 4 to solve a problem in L, then either $\{x^k\}$ is finite and the last point is optimal or $\{x^k\}$ is infinite and contains cluster points, each of which is optimal. Furthermore, when a problem in L is such that $f^0(.)$ is strictly convex, then that problem has a unique optimal solution x^*, and in that case $\{x^k\}$converges to x^*.

We now give a slight modification to Algorithm 4 which allows the resulting algorithm to achieve finite convegence when used to solve problems with quadratic cost.

Algorithm 5: Let x^1 and $\epsilon > 0$ be given.

Step 1: Use a conjugate gradient method to find the minimizer x_u^* for the unconstrained version of the problem.

Step 2: If $x_u^* \in \Omega$ then stop; else go to Step 3.

Step 3: Compute x_p^1 to be the projection of x_u^* onto the set Ω.

Step 4: Use Algorithm 4 with x_p^1 as the starting point to compute the solution of the problem.

Corollary 1: If $\{x^k\}$ is a sequence constructed by Algorithm 5 to solve a problem in L with $f^0(.)$ a quadratic in x with postitive definite Hessian, then $\{x^k\}$ is finite and the last point is optimal.

V. DISCRETE LQR CONTROL PROBLEM WITH HARD CONTROL BOUNDS

In this section we rewrite the discrete LQR optimal control problem with hard control bounds as a bounded variable quadratic programming problem and then show how the gradient and step length computations may be organized to share common terms.

Problem 3: Given an m-input, discrete, time-varying linear system in which we are given the initial state, $z_0 \in E^n$, and

$$z_i = A_i z_{i-1} + B_i u_i, \quad i = 1, 2, ..., N, \tag{5.1}$$

where for $i = 0, 1, ..., N$, $z_i \in E^n$ is the state of the system at time i and for $i = 1, 2, ..., N$, $u_i \in E^m$ is the control at time i with components $u_{i,j}$, $j = 1, 2, ..., m$, find the mN control vector $u = (u_1^t, u_2^t, \ldots, u_N^t)^t$ that minimizes the performance index

$$J(u) = \frac{1}{2}\sum_{i=1}^{N}\left(z_i^t Q_i z_i + u_i^t R_i u_i\right),$$

and satisfies

$$u \in \Omega,$$

where for $i = 1, 2, \ldots, N$, Q_i are $n \times n$ symmetric positive semi-definite matrices, R_i are $m \times m$ symmetric positive definite matrices and the convex compact set Ω is defined as

$$\Omega = \{u \mid \ |u_{i,j}| \le 1, \ \text{for } i = 1, 2, \ldots, N, \ j = 1, 2, \ldots, m\}.$$

Using the notation introduced in [MEY86], Problem 3 may be rewritten as a bounded variable quadratic programming problem in u:

Problem 4: Find the control vector $u \in E^{mN}$ that minimizes the performance index $J(u) = 1/2\, u^t H u + b^t u + c$ subject to $u \in \Omega$, where H is the $mN \times mN$ block symmetric positive definite matrix with block size $m \times m$ given by $H = (R + F_u^t F_z^{-t} Q F_z^{-1} F_u)$, b is the $mN \times 1$ vector given by $b^t = z_0^t F_0^t F_z^{-t} Q F_z^{-1} F_u$, and c is the scalar given by $c = 1/2 z_0^t F_0^t F_z^{-t} Q F_z^{-1} F_0 z_0$, and the matrices R, Q, F_z, F_u and F_0 are defined as: $Q = Diag(Q_1, Q_2, \ldots, Q_N)$, is the $nN \times nN$ block matrix that consists of N $n \times n$ symmetric positive semi-definite blocks Q_i, $R = Diag(R_1, R_2, \ldots, R_N)$ is the $mN \times mN$ block matrix that consists of N $m \times m$ symmetric positive definite blocks R_i, F_z is the $nN \times nN$ block lower bidiagonal matrix that consists of N^2 $n \times n$ blocks $\left(F_z\right)_{ij}$ and F_u is the $nN \times mN$ block diagonal matrix that consists of N^2 $n \times m$ blocks $\left(F_u\right)_{ij}$ defined for all i and j in $[1,2,\ldots,N]$ by

$$\left(F_z\right)_{ij} = \begin{cases} I & \text{if } i = j \\ -A_i & \text{if } i = j+1, \\ 0 & \text{otherwise} \end{cases} \qquad \left(F_u\right)_{ij} = \begin{cases} B_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

and F_0 is the $nN \times n$ block matrix that consists of N $n \times n$ blocks $\left(F_0\right)_i$ defined for all i in $[1,2,\ldots,N]$ by

$$\left(F_0\right)_i = \begin{cases} A_1 & \text{if } i = 1 \\ 0 & \text{otherwise}. \end{cases}$$

Since Problem 4 is of the form of Problem 2, Algorithms 3 and 4 can be used to solve Problem 4. We now use the approach for the evaluation of $g(u^1)$, α^k and $g(u^{k+1})$ given in [MEY87b] in which α^k and $g(u^{k+1})$ are computed by sharing common terms, where we use the notation $g(u^k) = (dJ/du)^t_{u=u^k}$.

We first consider the computation of $g(u^1)$, which is performed only once in Step 0 of Algorithms 3 and 4. The matrix F_z is non-singular and thus we may write the gradient as $g(u^1) = Ru^1 + F_u^t F_z^{-t} Q z^1$, where z^1 satisfies Eq. (5.1). Let $\lambda \in E^{nN}$ be the costate vector $\lambda = (\lambda_1^t, \lambda_2^t, \ldots, \lambda_N^t)^t$, $\lambda_i \in E^n$, defined by $\lambda = F_z^{-t} Q z$. Then, given u^1 and z_0, the gradient $g(u^1)$ may be obtained by using the three equations

$$F_z z^1 = F_u u^1 + F_0 z_0, \tag{5.2}$$

$$F_z^t \lambda^k = Q z^k, \tag{5.3}$$

$$g(u^1) = Ru^1 + F_u^t \lambda^1. \tag{5.4}$$

As a consequence of Eqs. (5.2), (5.3) and (5.4) we obtain the gradient evaluation technique proposed by Polak [POL71]:

Algorithm 6: Given u^1 and z_0.

Step 1: Compute z^1 such that $F_z z^1 = F_u u^1 + F_0 z_0$.

Step 2: Compute λ^1 such that $F_z^t \lambda^1 = Q z^1$.

Step 3: Compute $g(u^1) = Ru^1 + F_u^t \lambda^1$.

Due to the lower $n \times n$ block bidiagonal structure of F_z, Steps 1 and 2 of Algorithm 6 require procedures for the solution of N stage forward and reverse $n \times n$ block first-order linear recurrences, respectively. Such procedures are discussed in Section VI. Given λ^1, Step 3 computes $g(u^1)$ by computing each of the N uncoupled components $g_i^1 = (dJ/du_i)^t_{u=u^1}$; thus, Step 3 exhibits linear speedup when executed in parallel.

Algorithms 3 and 4 both use the common term $\pi^k = Hd^k$ in the computation of the step length χ^k and the next gradient $g(u^{k+1})$. We now show that π^k can be efficiently computed by writing π^k as

$$\pi^k = Rd^k + F_u^t F_s^{-t} Q F_s^{-1} F_u d^k,$$

and using Algorithm 7 below, where we note that instead of requiring F_s^{-1} and F_s^{-t}, we solve linear systems corresponding to F_s and F_s^t.

Algorithm 7: Given d^k.

Step 1: Compute $\mu^k = Rd^k$.

Step 2: Compute $\delta^k = F_u d^k$.

Step 3: Compute w^k such that $F_s w^k = \delta^k$.

Step 4: Compute $\eta^k = Qw^k$.

Step 5: Compute v^k such that $F_s^t v^k = \eta^k$.

Step 6: Compute $\chi^k = F_u^t v^k$.

Step 7: Compute $\pi^k = \mu^k + \chi^k$.

With the exception of Steps 3 and 5, Algorithm 7 computes π^k by executing a series of matrix-vector products followed by a vector sum. Each of the matrix-vector products consists of N uncoupled block matrix-vector products, thus exhibiting linear speedup when implemented in parallel. However, due to the structure of F_s, Steps 3 and 5 require the solution of N stage forward and reverse $n \times n$ block first-order linear recurrences, respectively. As in the case of Algorithm 6, this again suggests the need for parallel procedures to solve linear recurrences.

VI. PARALLEL ALGORITHMS FOR OPTIMAL CONTROL PROBLEMS

The model of SIMD parallel computation that we use consists of a global parallel memory, p parallel processors, and a control unit, where all processors perform the same operation at each time step. We further simplify the model by making the following assumptions:

A1. Each computational operation requires the same amount of time, referred to as a step.

A2. There are no accessing conflicts in global memory.

A3. All initial data resides in global memory.

A4. There is no time required to access global memory.

We use the parallel procedures FORWARD and REVERSE given in [MEY87b] to solve the forward and reverse N stage $n \times n$ block first-order linear recurrence systems that are required by Algorithms 6 and 7. The procedures are blocked versions of the parallel scalar approach given in [MEY87a] and are formulated as a function of the number p of processors so that the algorithm structure is fixed whenever the number of stages $N \geq (p/n)^2$. We then use the parallel procedures GRADIENT and DIRECTION also given in [MEY87b] to obtain parallel implementations of Algorithms 6 and 7.

We next give the following parallel implementation of computing the quanities $p^k = w(u^k, I(u^k,\epsilon)) - u^k$ and $\hat{p}^k = -P_k g(u^k)$.

```
1. PROCEDURE PROJECTION1($u^k$, $g(u^k)$)
2.  FORALL $i \in \{1,2,...,N\}, j \in \{1,2,...,m\}$ DO IN PARALLEL $p_{i,j}^k := -g_{i,j}(u^k)$;
3.  FORALL $i \in \{1,2,...,N\}, j \in \{1,2,...,m\}$ DO IN PARALLEL
4.    IF $u_{i,j}^k - 1 \geq -\epsilon$ AND $u_{i,j}^k - g_{i,j}(u^k) > 1$ THEN $p_{i,j}^k := +1 - u_{i,j}^k$;
5.    IF $-u_{i,j}^k - 1 \geq -\epsilon$ AND $u_{i,j}^k - g_{i,j}(u^k) < -1$ THEN $p_{i,j}^k := -1 - u_{i,j}^k$;
6.  END FORALL
7.  RETURN $p^k$;
8. END PROCEDURE
```

```
1. PROCEDURE PROJECTION2($u^k$, $g(u^k)$)
2.   FORALL $i \in \{ \quad ,\ldots,N\}$, $j \in \{1,2,\ldots,m\}$ DO IN PARALLEL $p \quad := g_{i,j}(u^k)$;
3.   $I(u^k,0) := \phi$;
4.   FORALL $i \in \{1,2,\ldots,N\}$, $j \in \{1,2,\ldots,m\}$ DO IN PARALLEL
5.     IF $u^k_{i,j} - 1 = 0$ THEN $\hat{p}^k := 0$, $I(u^k,0) := I(u^k,0) \cup (i,j)$;
6.     IF $-u^k_{i,j} - 1 = 0$ THEN $\hat{p}^k := 0$, $I(u^k,0) := I(u^k,0) \cup (i,j)$;
7.   END FORALL
8.   RETURN $\hat{p}^k$, $I(u^k,0)$;
9. END PROCEDURE
```

We now embed the parallel procedures GRADIENT, DIRECTION, and PROJECTION1 to obtain a parallel implementation of Algorithm 3 to solve Problem 4 and we then give the corresponding number of steps required for one iteration using p processors.

```
1. PROCEDURE PGPM($z_0$,$u^1$)
2.   $k := 1$;
3.   $g(u^1) :=$ GRADIENT($z_0$,$u^1$);
4.   $p^1 :=$ PROJECTION1($u^1$, $g(u^1)$);
5.   WHILE $\|p^k\|^2 > \varepsilon_s$ DO
6.     $d^k := p^k$;
7.     $\pi^k :=$ DIRECTION($d^k$);
8.     $\alpha^k := \min\{\alpha_s, \alpha_c\}$;
9.     FORALL $i \in \{1,2,\ldots,N\}$ DO IN PARALLEL $u_i^{k+1} := u_i^k + \alpha^k d_i^k$;
10.    FORALL $i \in \{1,2,\ldots,N\}$ DO IN PARALLEL $g_i^{k+1} := g_i^k + \alpha^k \pi_i^k$;
11.    $p^{k+1} :=$ PROJECTION1($u^{k+1}$, $g(u^{k+1})$);
12.    $k := k + 1$;
14.  END WHILE
15. END PROCEDURE
```

Theorem 4: *Given z_0, u^1, N, n, m and p such that $p = 1$ or p/n is an integer, the number of steps required* by one iteration of procedure PGPM using p processors, $1 \le p \le nN^{1/n}$, is

$$T(N,n,m,p) = \begin{cases} \left\lceil \frac{Nn}{p} \right\rceil (6n + 2m - 2) + \left\lceil \frac{Nm}{p} \right\rceil (2n + 2m + 22) - \frac{4n^2}{p} + 4\log_2 p & \text{if } 1 \le p \le n \\ \left\lceil \frac{Nn}{p} \right\rceil (2n + 2m - 2) + \left\lceil \frac{Nm}{p} \right\rceil (2n + 2m + 22) + \left\lceil \frac{N-1}{(p/n)^2 - 1} \right\rceil 8(p-n) + 4\log_2 p & \text{if } n < p \le nN^{1/n}. \end{cases}$$

We next embed the parallel procedures GRADIENT, DIRECTION, PROJECTION1 and PROJECTION2 to obtain the following parallel implementation of Algorithm 4 to solve Problem 4 and we then give the corresponding number of steps required for one iteration.

```
1. PROCEDURE PGPFRM($z_0$,$u^1$)
2.   $k := 1$;
3.   $g(u^1) :=$ GRADIENT($z_0$,$u^1$);
4.   $p^1 :=$ PROJECTION1($u^1$, $g(u^1)$);
5.   WHILE $\|p^k\|^2 > \varepsilon_s$ DO
6.     IF $d^{k-1} = 0$ THEN $d^k := p^k$
7.       ELSE $\beta^k = \|p^k\|^2 / \|p^{k-1}\|^2$
             FORALL $i \in \{1,2,\ldots,N\}$ DO IN PARALLEL $d_i^k := p_i^k + \beta^k d_i^{k-1}$;
8.     END IF
9.     $\pi^k :=$ DIRECTION($d^k$);
10.    $\alpha^k := \min\{\alpha_s, \alpha_o\}$;
11.    FORALL $i \in \{1,2,\ldots,N\}$ DO IN PARALLEL $u_i^{k+1} := u_i^k + \alpha^k d_i^k$;
12.    FORALL $i \in \{1,2,\ldots,N\}$ DO IN PARALLEL $g_i^{k+1} := g_i^k + \alpha^k \pi_i^k$;
13.    $p^{k+1} :=$ PROJECTION1($u^{k+1}$, $g(u^k) + 1$);
```

```
14.   $\hat{p}^{k+1}$ := PROJECTION2($u^{k+1}, g(u^{k+1})$);
15.   IF $I(u^{k+1},0)$ [illegible] $^{k},0)$ OR $p^{k+1} \neq \hat{p}^{k+1}$ THEN $d^k := 0$;
16.   $k := k + 1$;
17. END WHILE
18. END PROCEDURE
```

Theorem 5: Given z_0, u^1, N, n, m and p such that $p = 1$ or p/n is an integer, the number of steps required by one iteration of procedure PGPFRM using p processors, $1 \leq p \leq nN^{1/2}$, is

$$T(N,n,m,p) = \begin{cases} \left\lceil \frac{Nn}{p} \right\rceil (6n + 2m - 2) + \left\lceil \frac{Nm}{p} \right\rceil (2n + 2m + 34) - \frac{4n^2}{p} + 7\log_2 p & \text{if } 1 \leq p \leq n \\ \left\lceil \frac{Nn}{p} \right\rceil (2n + 2m - 2) + \left\lceil \frac{Nm}{p} \right\rceil (2n + 2m + 34) + \left\lceil \frac{N-1}{(p/n)^2 - 1} \right\rceil 8(p-n) + 7\log_2 p & \text{if } n < p \leq nN^{1/2}. \end{cases}$$

The speedup S_p and efficiency E_p for procedures PGPM and PGPFRM can be obtained directly from Theorems 4 and 5. A straightforward computation shows that S_p and E_p are bounded from below by $S_p \geq 0.6p$ and $E_p \geq 0.6$.

VII. CONCLUSIONS

In this paper two parallel algorithms have been presented to solve the discrete LQR optimal control problem with hard control bounds. The algorithms exhibit fast convergence coupled with a high degree of parallelism at each iteration, making them suitable for real-time online implemenation on an SIMD machine. Moreover, the algorithms possess the desirable property that their structure, and hence parallelism, is determined by the number of available processors. Thus, unlike approaches in which the structure of the procedure changes with problem size, the procedures presented in this paper maintain the same computational and interprocessor communication requirements independently of the number of stages in the control problem. Although not considered in this paper, interprocessor communication requirements should not be a critical performance factor in view of the implementation results for parallel linear recurrence solvers presented in [MEY87a].

Finally, we note that the parallel procedures presented in this paper may be used to solve constrained discrete optimal control problems which involve nonquadratic cost and nonlinear dynamics. In that case one uses suitable approximations in which the system dynamics are linearized and the cost is approximated quadratically.

REFERENCES

[MEY73] Meyer, G.G.L., A Segmented Algorithm for Solving a Class of State Constrained Discrete Optimal Control Problems, *Decision and Control Theory Conference*, San Diego, California, 1973, pp. 73-79.

[MEY87a] Podrazik, L.J. and Meyer, G.G.L., A Parallel First-Order Linear Recurrence Solver, *Journal of Parallel and Distributed Computing*, Vol. 4, No. 2, April, 1987, pp. 117-132.

[MEY87b] Podrazik, L.J. and Meyer, G.G.L., Parallel Implementations of Gradient Based Iterative Algorithms for a Class of Discrete Optimal Control Problems, *1987 International Conference on Parallel Processing*, August, 1987, pp. 491-494.

[POL71] Polak, E.J., *Computational Methods in Optimization: A Unified Approach*, Academic Press, New York, New York, 1971.

[SCH81] Scheel, C. and McInnis, B., Parallel Processing of Optimal Control Problems by Dynamic Programming, *Information Sciences*, Vol. 25, No. 2, November 1981, pp. 85-114.

[TRA80] Travassos, R. and Kaufman, H., Parallel Algorithms for Solving Nonlinear Two-Point Boundary-Value Problems Which Arise in Optimal Control, *Journal of Optimization Theory and Applications*, Vol. 30, No. 1, January 1980, pp. 53-71.